НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

На правах рукопису
УДК 004.8

До захисту допущено
В. о. завідувача кафедри ММСА
О.Л.Тимощук
«___» _____ 2019 р.

Магістерська дисертація
на здобуття ступеня магістра за спеціальністю 124 Системний аналіз
на тему: «Методи тривимірної реконструкції моделей об'єкта з послідовності фотозображень»

Виконав:
студент II курсу, групи КА-81мп
Скришевський Остап Едуардович _____

Керівник: професор кафедри ММСА
д.т.н. проф. Данилов Валерій Яковлевич _____

Рецензент: професор кафедри інформаційної безпеки
КПІ ім. Ігоря Сікорського
д.т.н. проф. Качинський А. Б. _____

Засвідчую, що у цій магістерській дисертації
немає запозичень з праць інших авторів
без відповідних посилань
Студент _____

Київ
2019

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

Рівень вищої освіти — другий (магістерський)
Спеціальність — 124 «Системний аналіз»

ЗАТВЕРДЖУЮ
В. о. завідувача кафедри ММСА
О. Л. Тимощук
«___»_____ 2019 р.

ЗАВДАННЯ

на магістерську дисертацію студенту Скришевському Остапу Едуардовичу

1. Тема дисертації: «Методи тривимірної реконструкції моделей об'єкта з послідовності фотозображень», науковий керівник дисертації Данилов Валерій Яковлевич, д.н.т, затверджені наказом по університету від «08» листопада 2019 р. № 3862-С.

2. Термін подання студентом дисертації: 13 грудня 2019 р.

3. Об’єкт дослідження: методи тривимірної реконструкції моделей.

4. Предмет дослідження: використання згорткових нейронних мереж для ідентифікації особливих точок у задачах тривимірної реконструкції

5. Перелік завдань, які потрібно виконати:

1) дослідити сучасні методи та алгоритми застосування згорткових нейронних мереж та алгоритмів для ідентифікаії та зіставлення особливих точок;

2) розробити згорткову нейронну мережу для ідентифікації особливих точок;

3) розробити метод реконструкції обєкту з послідовності фотографій;

4) на базі розробленого методу створити програмний модуль;

5) розробити стартап-проект виведення на ринок результатів дослідження;

6) розробити висновки за результатами наукового дослідження.

6. Дата видачі завдання: 05 вересня 2019 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації
1.	Концептуальний вступ дисертації. Формулювання об’єкта, предмета, цілі, завдань, новизни, практичної значущості результатів	18.09.2019—20.09.2019
2.	Перший розділ. Огляд літературно-інформаційних джерел. Понятійно-категоріальний апарат. Характеристика об’єкта	21.09.2019—30.03.2019
3.	Другий розділ. Імплементація модуля LIFT для виявленная особливих точок з фото	31.03.2019—16.04.2019
4.	Третій розділ. Розробка базового методу на базі згорткової нейроної мережі	17.04.2019—25.04.2019
5.	Четвертий розділ. Стартап-проект	25.04.2019—06.05.2019
6.	Концептуальні висновки. Перспективи розвитку отриманих рішень	07.05.2019—10.05.2019

Студент О. Е. Скришевський

Науковий керівник дисертації В. Я. Данилов

РЕФЕРАТ

Магістерська дисертація: 102 с., 22 рис., 28 табл., 1 додаток, 29 джерел.

ШТУЧНІ НЕЙРОННІ МЕРЕЖІ, ГЛИБОКЕ НАВЧАННЯ, ВИЯВЛЕННЯ ОБ'ЄКТІВ, РЕЖИМ РЕАЛЬНОГО ЧАСУ, OBJECT DETECTION.

Об'єкт дослідження - методи тривимірної реконструкції моделей.

Предмет дослідження - використання модуля ідентифікації особливих точок на базі згорткових нейронних мереж у задачах тривимірної реконструкції.

Мета роботи - дослідити сучасні методи реконструкції об'єктів у тривимірному просторі.

З ціллю проведення дослідження було розроблено програмний модуль, призначену для тривимірної реконструкції будинку з використання системи ідентифікації особливих точок на базі згорткових нейронних мереж. В роботі досліджено наступні методи: SIFT та LIFT та їх модифіувції. Обрані методи досліджувалися з використанням opensource набору даних.

Результатами роботи є розроблений модуль, отриманий в результаті дослідження тривимірної реконструкції обєктів, показники ефективності отриманого методу та напрямки для подальших досліджень.

Наукова новизна полягає в виділенні ідей та напрямків подальшого дослідження з метою отримання нових рішень в галузі фотографметрії.

ABSTRACT

Master’s thesis: 102 p., 22 pic., 28 tables, 1 attachment, 29 sources.

ARTIFICIAL NEURAL NETWORKS, DEEP LEARNING, OBJECT DETECTION, FACE DETECTION, MOBILE DEVICES, REALTIME, TENSORFLOW, TENSORFLOW OBJECT DETECTION API, GOOGLE CLOUD ML ENGINE.

Object of study - methods of three-dimensional reconstruction of models.

The subject of the study is the use of the module of identification of singular points based on convolutional neural networks in the problems of three-dimensional reconstruction.

The purpose of the work is to explore modern methods of reconstruction of objects in three-dimensional space.

For the purpose of the study, a software module was developed designed for three-dimensional home reconstruction using a system of singular point identification based on convolutional neural networks. The following methods are investigated in the paper: SIFT and LIFT. Selected methods were investigated using the opensource dataset.

The results of the work are a developed module, obtained as a result of the study of three-dimensional reconstruction of objects, performance indicators of the obtained method and directions for further research.

Scientific novelty is to identify ideas and directions for further research in order to obtain new solutions in the field of photometry.

ЗМІСТ

ВСТУП 8
РОЗДІЛ 1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ 9
1.1 Алгоритми відновлення структури сцени по фокусуванні 9
1.2 Алгоритми другого типу засновані на обчисленні відстані до всіх точок зображення. 10
1.3 Реконструкція за допомогою проектування 12
1.4 Реконструкція за допомогою 3D-сканера 14
1.6 3D-реконструкція за допомогою набору зображень об'єкта 15
Висновки до розділу 17
РОЗДІЛ 2 МЕТОДИ ТРИВИМІРНОІ РЕКОНСТРУКЦІІ МОДЕЛІ 18
2.1 Камера. Модель проективної камери 18
2.1.1 Внутрішнє калібрування камери 21
2.1.2 Матриці пари камер. Калібрування 22
2.2 Епіполярна геометрія 23
2.3 Особливі точки і дескриптори 26
2.4 Зіставлення ключових точок 28
2.6 Оцінка фундаментальної матриці 31
2.7 Нормалізований 8-точковий алгоритм 32
2.8 Відновлення відносної позиції камер 35
2.8.1 Обчислення базової матриці 35
2.8.2 Відносна позиція основної матриці 36
2.9 Реконструкція великогабаритної моделі 39
РОЗДІЛ 3 РЕЗУЛЬТАТ РОБОТИ ПРОГРАМНОГО ПРОДУКТУ 48
РОЗДІЛ 4 РОЗРОБКА СТАРТАП-ПРОЕКТУ 54
4.1 Опис ідеї та технологічний аудит стартап-проекту 54
4.2 Технологічний аудит ідеї стартап-проекту 56
4.3 Аналіз ринкових можливостей запуску стартап-проекту 57
4.4 Розроблення ринкової стратегії проекту 66
4.5 Розроблення маркетингової програми стартап-проекту 70

Висновки до розділу 74
ВИСНОВКИ 75
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 76
ДОДАТОК А 79

ВСТУП

Відновлення тривимірної моделі сцени з набору зображень - класична задача комп'ютерного зору, в якій тривимірна модель сцени повинна бути відновлена з набору фотографій цієї сцени.

Розвиток віртуальної реальності, систем управління транспортними засобами, візуалізації на основі аналізу зображень, медичної промисловості, а також інших областей, які потребують побудови тривимірних моделей, призвело до збільшення уваги до цієї області. Найбільшим інтересом користуються випадки, в яких положення камер не відомі і / або можуть змінюватися з плином часу. Так, наприклад, в системах активного зору калібрувальні параметри можуть змінюватися під час роботи системи.

Звичайно, існують лазерні 3D сканери, що дозволяють створювати дуже точну тривимірну модель сцени, але їм властиво безліч недоліків. У їх числі дуже висока вартість подібної апаратури, вельми обмежені розміри сканованих об'єктів, низька продуктивність і багато іншого. Більш того, такі пристрої не зовсім безпечні для людини, зокрема для очей.

На даний момент одними з найпопулярніших бібліотек для роботи з алгоритмами комп'ютерного зору є OpenCV (Open Source Computer Vision Library) і PCL (Point Cloud Library). Бібліотеки Кросплатформені, реалізовані на мові високого рівня (C/C ++), але розробляються і для інших мов програмування. Поширюються в умовах ліцензії BSD, отже, можуть бути використані як в наукових, так і в комерційних цілях.

РОЗДІЛ 1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

На сьогоднішній день пропонується цілий ряд рішень завдання побудови 3D-моделі об'єктів. Деякі з них передбачають використання спеціального обладнання, наприклад, 3D-сканерів, інші ж можуть обмежитися можливостями веб-камер. Основна частина методів полягає в знаходженні розташування точок об'єкта, їх координат в координатному просторі без зміни масштабу, і в разі потреби, в накладенні текстур на точки моделі [2].

Для отримання результату високої якості незалежно від обраного методу (за деяким винятком), необхідне виконання деяких умов [11]:

1. Висока якість вхідних даних.

2. Однакове освітлення поверхонь при зміні кута огляду.

3. Калібрування камер (вивчення їх технічних параметрів і розташування в просторі).

4. Існування областей перетинів у зображень.

5. Різка зміна інтенсивності на кордоні об'єкта.

1.1 Алгоритми відновлення структури сцени по фокусуванні

Дані алгоритми побудови моделі сцени засновані на тому, що глибина різкості оптичних систем кінцева. При управлінні параметрами фокусування камери можна досягти різкого зображення різних точок об'єкта, а вже знаючи параметри фокусування камери при яких досягається потрібна резкозть можна оцінити глибину цих точок.

В [7] описаний алгоритм реконструкції сцени по фокусуванні, що володіє великою точністю. В роботі досліджено порядок точності алгоритму,

а також методичні похибки, однак, автори акцентують увагу на тому, що збіжність даного методу не доведена, що, взагалі кажучи, є типовим для подібних завдань.

1.2 Алгоритми другого типу засновані на обчисленні відстані до всіх точок зображення.

Серед змішаних методів найбільший інтерес представляє алгоритм Бернарда [9], основна ідея якого полягає в побудові карти зміщення, яка будується на основі обчислення зсуву в кожній точці зображення. Однак цей алгоритм не ефективний при недостатній кількості інформації про просторову структуру.

В [12] описаний метод, який дозволяє об'єднати алгоритми відновлення по замальовці і по стереопарі для більш точного виявлення ребер, меж та інших великих деталей.

Алгоритми відновлення структури сцени по набору зображень, отриманих при русі спостерігача.

В даний час, алгоритми цього класу користуються найбільшою популярністю [13-15]. Тут передбачається, що зображення, за якими реконструюється сцена, зняті з різних позицій і можливо в різний час, при цьому часто передбачається, що невідомі положення камери і невідомі або постійно змінюються її внутрішні параметри. Загальний процес реконструкції моделі сцени в рамках цієї групи алгоритмів складається з двох етапів:

1) Пошук особливих точок: на зображеннях здійснюється пошук локальних особливостей у вигляді точок, ліній, а також шукаються відповід ності між знайденими про собенно ня на послідовності зображень.

2) Обчислення структури: для знайдених відповідностей обчислюються їх позиції в тривимірному просторі за принципом тріангуляції. На отриманий об'єкт натягується поверхню.

Вхідними даними для алгоритмів цього типу є координати особливих точок на зображеннях і особливі ідентифікатори відповідні їм, завдяки яким кожну особливу точку можна відрізнити від дугіх, причому, на всій колекції зображень кожній точці, являющейсяся проекцією деякої точки на тривимірній сцені повинен відповідати однаковий індентіфікатор. Безпосередньо з зображеннями алгоритми даної групи не працюють.

Останнім часом, велику увагу дослідників привертають методи, в основі яких лежить ідея факторизации матриць [16-19]. В [16], [18] детально описана основна ідея таких методів. В [17] розглядається випадок, коли на сцені присутні кілька, можливо рухомих, об'єктів, а також пропонується метод підрахунку і розподілу таких об'єктів. В [19] описана модифікація для додатків реального часу. Модель будується за наявними даними, а при надходженні нових даних модель уточнюється. При такому підході не обов'язково мати всі зображення сцени на момент початку побудови тривимірної моделі і на практиці потрібно менше обчислювальних ресурсів. Однак, в цьому випадку на кожній ітерації доповнення і уточнення моделі сцени буде відбуватися накопичення похибок. Ця проблема поки ніяк не вирішена.

1.3 Реконструкція за допомогою проектування

Даний метод передбачає розробку образу бажаного об'єкта вручну. Існує цілий ряд програм, що використовують даний метод, причому як в офлайн (Blender, Autodesk 3D max) (рис. 1.1), так і в онлайн (Tinkercad, Sketchfab) (рис. 1.2) варіантах. При створенні моделі об'єкта проектувальник будує його образ на основі свого сприйняття, пряму взаємодію з зображеннями об'єкта відсутня. Таким чином за допомогою даного методу можна будувати 3D-моделі як конкретних об'єктів (будівлі, меблі), так і об'єктів, які існують лише в уяві проектувальника (різні абстракції).

Етапи побудови 3D-моделі:

1. Побудова полігональної моделі, званої каркасом. Досягається шляхом відновлення геометричної форми і розмірів бажаного об'єкта.

2. Вибір становить матеріалу. Наприклад, застосування до будівлі при його проектуванні текстури дерева.

3. Налаштування освітлення.

4. Налаштування сцени: розташування камер для спостерігача.

5. Створення зовнішніх впливів (при необхідності).

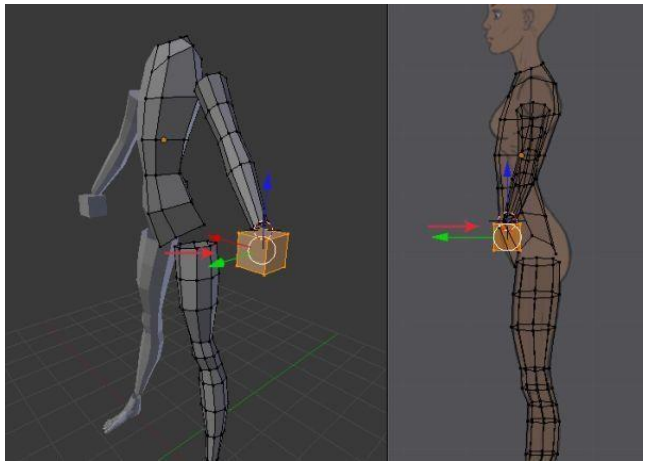


Рисунок 1.1 - Процес моделювання тіла людини з використанням Blender

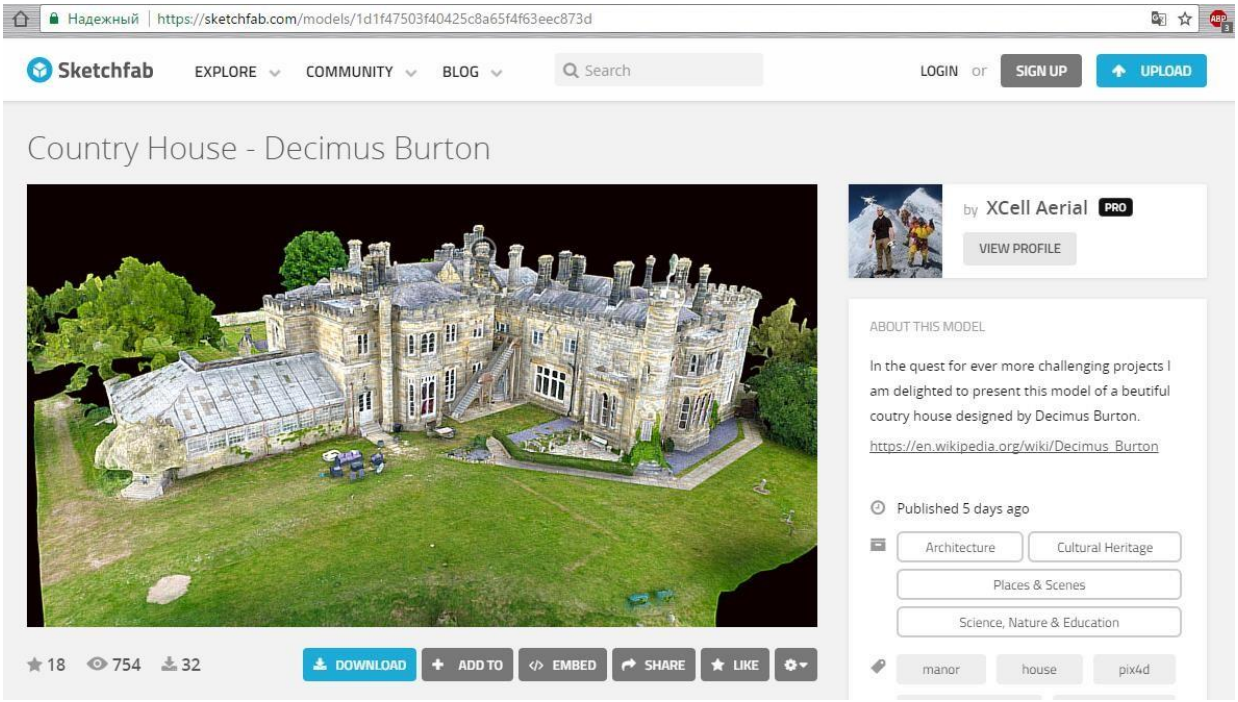


Рисунок 1.2 - Процес моделювання архітектури з використанням Sketchfab

Даний метод має безліч переваг. Він має високу точність, що дозволяє проектувальнику через відображення найменших деталей і розмірів здійснювати побудову моделей будь-яких об'єктів. Однак, для отримання задовільного результату потрібно багато часу, що, безсумнівно, є істотним недоліком методу.

1.4 Реконструкція за допомогою 3D-сканера

3D-сканер - периферійний пристрій, що аналізує об'єкт і на основі отриманих даних створює його 3D-модель.

За методом сканування вони діляться на два типи: контактний і безконтактний. Перший передбачає використання спеціального щупа, який досліджує точки, вибрані оператором. Точність одержуваної інформації, залежить від датчиків, розташованих на щупі, які схожі з датчиками оптико-механічною миші. Даний тип сканерів передбачає роботу з об'єктами невеликого розміру і простої форми. Безконтактні 3D-сканери мають більш складну структуру і оперують ультразвуком, різними лазерними датчиками і фотокамерами. Відбувається комбінування даних самих зображень і результатів, отриманих від інших використовуваних датчиків.

Яскравим прикладом служить сканування, що використовує білий світ (рис. 1.4). Сканер направляє на об'єкт шаблон закодованого світла від джерела, який і «привласнює» форму об'єкту. Дані узгоджуються зі знімками і утворюють якісну модель.

Даний метод дозволяє отримати найточнішу 3D-модель за короткий проміжок часу. Однак дані технічні установки не тільки дорогі, а й стаціонарні, що не дозволяє реконструювати будь-який об'єкт

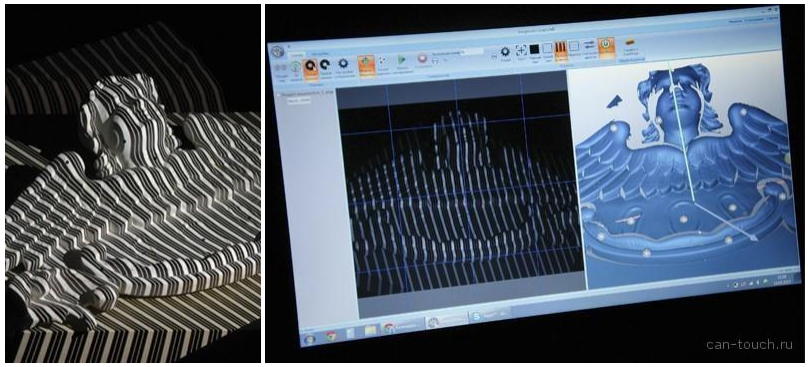


Рисунок 1.4 - Сканування білим світлом

1.5 Реконструкція за допомогою стереопари

Метод оперує набором лише з двох зображень, отриманих за допомогою стереокамери і є, можна сказати, підзадачею технології, описаної в п.1.2. В даному випадку аналогічно здійснюється вибір точок відповідності, їх зіставлення і геометричні перетворення [10, 11].

Таким чином, для отримання 3D-моделі необхідно виконання наступного алгоритму дій:

1. Визначення фундаментальної матриці.
2. Визначення матриці камер.
3. Знаходження відповідних точок.
4. Побудова хмари точок.
5. Текстурування.

Однак, модель, отримана подібним чином, не рахується «Повним» 3D-образом, так як будується лише поверхневий «вигляд» об'єкта. Привертає ж до себе даний метод своїми швидкістю і доступністю

1.6 3D-реконструкція за допомогою набору зображень об'єкта

Метод використовує послідовний ряд зображень об'єкта (рис. 1.3). При цьому необхідний відсоток накладення двох суміжних кадрів повинен перевищувати 50%, а мінімальна кількість перекриваються кадрів має дорівнювати трьом. При грамотному виконанні цих умов можна отримати досить якісний образ, з подальшим редагуванням лише розміру об'єкта [8, 9]

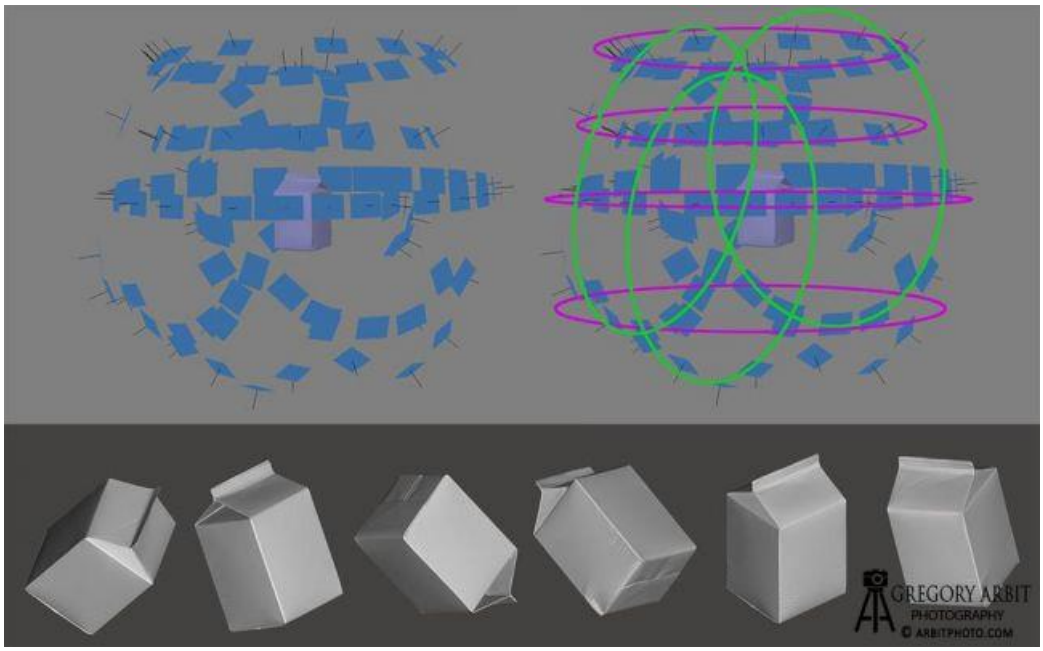


Рисунок 1.3. Загальна схема фотографування об'єкта

Алгоритм роботи даного методу можна представити таким переліком етапів [10]:

1. Фотографування досліджуваного об'єкта і імпорт отриманих зображень.

2. Пошук особливих точок і рішення системи рівнянь, отриманої на підставі безлічі даних точок.

3. Визначення параметрів камери.

4. Пошук «однакових» точок на різних наборах суміжних зображень об'єкта.

5. Обчислення координат точок щодо «базового» зображення об'єкта.

6. Приведення точок до системи координат, найбільш зручною для аналізу об'єкта і накладення структури.

В даному методі в якості недоліків можна виділити стаціонарне положення об'єкта і довгий процес зйомки. Однак, збільшення кількості фотографій призводить до більш якісних результатів.

Висновки до розділу

Таким чином, на основі проведеного аналізу можна стверджувати, що завдання 3D-реконструкції дуже актуальна і існує безліч способів її рішення в залежності від поставлених обмежень. Для вирішення завдання даної роботи був обраний метод, який використовує стереопари.

РОЗДІЛ 2 МЕТОДИ ТРИВИМІРНОІ РЕКОНСТРУКЦІІ МОДЕЛІ

2.1 Камера. Модель проективної камери

Модель камери охоплює як внутрішні, так і зовнішні параметри калібрування. Вбудована калібрування камери може включати параметри, що описують спотворення об'єктиву [23]. Часто передбачається, що спотворення об'єктиву, присутні на вхідних зображеннях, незначні, і тому спотворення об'єктиву не є частиною моделі. Якщо на вхідних зображеннях присутня значна спотворення об'єктиву, будь-який відомий метод корекції може бути застосований до зображень до їх подачі в систему [17].

Сучасні CCD-камери добре описуються за допомогою наступної моделі, званої проективної камерою (projective camera, pinhole camera). Проективна камера визначається центром камери, головною віссю - променем починається в центрі камери і спрямованим туди, куди камера дивиться, площиною зображення – площиною, на яку виконується проектування точок, і системою координат на цій площині. У такій моделі, довільна точка простору X проектується на площину зображення в точку x лежить на відрізку CX, який з'єднує центр камери C з вихідною точкою X (рис. 2.1).

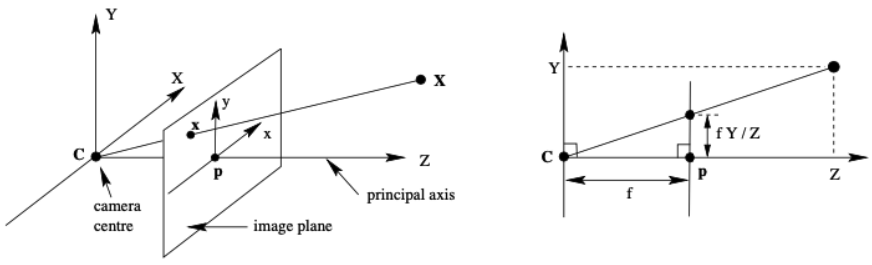


Рисунок 2.1 - Модель камери. C - центр камери, p - головна точка, Cp - головна вісь камери. Точка X тривимірного простору проектується в точку x - на площині зображення [23].

Формула проектування має просту математичну запис в однорідних координатах:

$$x = P X, \tag{2.1}$$

де X - однорідні координати точки простору, x - однорідні координати точки площини, P - матриця камери розміру 3×4.

Матриця P виражається наступним чином:

$$P = KR [I | -c] = K [R | t], \tag{2.2}$$

де K - верхня трикутна матриця внутрішніх параметрів камери розміру 3 × 3 (конкретний вид наведено нижче), R - ортогональна матриця розміру 3 × 3, яка визначає поворот камери щодо глобальної системи координат, I - одинична матриця розміру 3 × 3, вектор c - координати центру камери, а t = - Rc.

Варто зазначити, що матриця камери визначена з точністю до постійного ненульового множника, який не змінить результатів проектування точок по формулі x = (P X). Однак цей постійний множник зазвичай вибирається так, що б матриця камери мала вищеописаний вид.

У самому простому випадку, коли центр камери лежить на початку координат, головна вісь камери збігається осі Cz, осі координат на площині камери мають однаковий масштаб (що еквівалентно квадратним пікселям), а центр зображення має нульові координати. Матриця камери дорівнюватиме

$$P = K [I | 0] , \tag{2.3}$$

де

$$K = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.4}$$

У загальному випадку, початок координат в площині зображення не збігається з основною точкою, як це передбачалося в (2.4), тому відображення

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \mapsto \begin{bmatrix} fX/Z + p_x \\ fY/Z + p_y \end{bmatrix}, \tag{2.5}$$

де $[px\ py]^T$ - координати основної точки на площині зображення. В однорідних координатах це стає

$$\begin{bmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{bmatrix} = \begin{bmatrix} f & & p_x & 0 \\ & f & p_y & 0 \\ & & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}. \tag{2.6}$$

Введемо K в якості калібрувальної матриці камери

$$\mathrm{K} = \begin{bmatrix} f & & p_x \\ & f & p_y \\ & & 1 \end{bmatrix}, \tag{2.7}$$

тоді (2.1) можна записати як

$$\mathbf{x} = \mathrm{K}[\mathrm{I}\ \mathbf{0}]\mathbf{X}. \tag{2.8}$$

Для цифрових камер зображення формується на матриці і відображається у вигляді пікселів. Датчик може мати квадратні пікселі, і вимір координат зображення в пікселях додає додаткове нерівномірне масштабування. Нехай mx і my - це кількість пікселів на одиницю відстані в координатах зображення в напрямках x і y. Тепер матриця калібрування камери стає

$$\mathrm{K} = \begin{bmatrix} \alpha_x & & x_0 \\ & \alpha_y & y_0 \\ & & 1 \end{bmatrix}, \quad (2.9)$$

де $\alpha_x = f_{mx}$ і $\alpha_y = f_{my}$ є фокусна відстань камери з точки зору розмірів пікселів в напрямках x і y відповідно. Аналогічно, $[x_0\ y_0]^T$ - координати основної точки з точки зору розмірів пікселя, з $x_0 = m_x p_x$ і $y_0 = m_y p_y$.

У цій моделі, як і в звичайних цифрових фотоапаратах, передбачається, що не існує перекосу осей датчиків зображення. Таким чином, калібрувальна матриця камери K містить всі відповідні параметри калібрування, властиві камері.

2.1.1 Внутрішнє калібрування камери

Для відновлення структури і руху за допомогою процесу, описаного в розділі 2.2.3, внутрішні калібрувальні параметри для кожного з вхідних зображень повинні бути відомі заздалегідь або оцінені. У цьому розділі розглядаються два методи оцінки матриці калібрування камери K, наведеної в (2.9). Один метод аналізує зображення каліброваного шаблону, зроблене камерою, а інший використовує мета-інформацію, що міститься в файлах зображень, створених сучасними цифровими фотоапаратами.

Для зображень, отриманих за допомогою однієї і тієї ж камери з постійною фокусною відстанню, матриця калібрування камери не змінюється. Тому, якщо все вхідні зображення зроблені за допомогою однієї і тієї ж камери і без зміни рівня масштабування, можна розрахувати загальну величину K для всіх зображень. Цю внутрішню калібрування можна виконати, наприклад, шляхом послідовної зйомки зображень каліброваного шаблону з різних ракурсів. На основі зображень калібрувальної схеми можна

оцінити калибровочную матрицю камери за допомогою таких методів, як [17].

Хоча висока точність може бути досягнута за допомогою вищевказаного методу, в контексті даного методу цей варіант не розглядається. Сучасні цифрові камери зберігають велику кількість метаданих на додаток до отриманого зображенню при створенні файлу зображення. Ці метадані включають настройки камери, що використовуються при зйомці зображення, наприклад, діафрагму, витримку і фокусна відстань, а також статичну інформацію, таку як марка і модель камери. Ця інформація зберігається в форматі файлів змінних зображень (EXIF). Витягуючи фокусна відстань з даних EXIF, наявних у вхідних файлах зображень, можна отримати досить хорошу оцінку матриці калібрування камери.

2.1.2 Матриці пари камер. Калібрування

Нехай є дві камери, задані своїми матрицями P і P 'в деякій системі координат. У такому випадку говорять, що є пара відкаліброваних камер. Якщо центри камер не збігаються, то цю пару камер можна використовувати для визначення тривимірних координат спостережуваних точок.

Найчастіше, система координат вибирається так, що матриці камер мають вигляд P = K[I|0], P'= K'[R'|t']. Це завжди можна зробити, якщо вибрати початок координат збігається з центром першої камери, і направити вісь Z уздовж її оптичної осі.

Калібрування камер зазвичай виконується, за рахунок багаторазової зйомки деякого калібрувального шаблону, на зображенні можна легко виділити ключові точки, для яких відомі їх відносні положення в просторі.

Далі складаються і вирішуються системи рівнянь, що зв'язують координати проекцій, матриці камер і положення точок шаблону в просторі.

Існують загальнодоступні реалізації алгоритмів калібрування, наприклад, Matlab Calibration toolbox. Так само бібліотека OpenCV включає в себе алгоритми калібрування камер і пошуку каліброваного шаблону на зображенні.

2.2 Епіполярна геометрія

Нехай є дві камери (рис. 2.2). C - центр першої камери, C'- центр другий камери. Точка простору X проектується в x на площину зображення лівої камери і в x 'на площину зображення правої камери. Прообразом точки x на зображенні лівої камери є промінь xX. Цей промінь проектується на площину другої камери в пряму l ', звану епіполярной лінією. Образ точки X на площині зображення другої камери обов'язково лежить на епіполярной лінії l '.

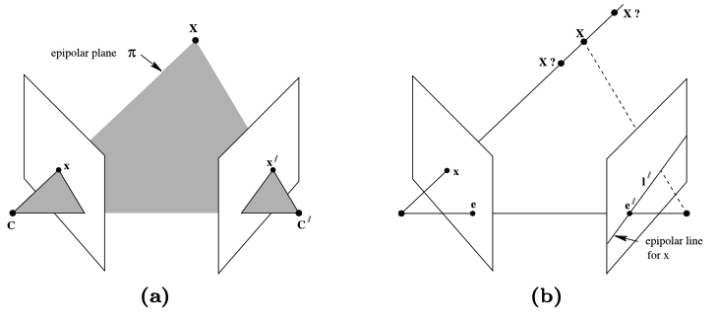


Рисунок 2.2 - Епіполярная геометрія

Таким чином, кожній точці x на зображенні лівої камери відповідає епіполярная лінія l 'на зображенні правою камери. При цьому пара для x на зображенні правою камери може лежати тільки на відповідній епіполярній

лінії. Аналогічно, кожній точці x 'на правому зображенні відповідає епіполярна лінія l на лівому.

Епіполярну геометрію використовують для пошуку стереопар, і для перевірки того, що пара точок може бути стереопарою (тобто проекцією деякої точки простору).

Епіполярна геометрія має простий запис в координатах. Нехай є пара відкаліброваних камер, і нехай x - однорідні координати точки на зображенні однієї камери, а x '- на зображенні другий. Існує така матриця F розміру 3 × 3, що пара точок x, x 'є стереопарою тоді і тільки тоді, коли: *x'TFx= 0*

Матриця F називається фундаментальною матрицею (fundamental matrix). Її ранг дорівнює 2, вона визначена з точністю до ненульового множника і залежить тільки від матриць вихідних камер P і P '.

У разі, коли матриці камер мають вигляд P = K[I|0], P'= K'[R|t] фундаментальна матриця може бути обчислена за формулою:

$$F = K'^{-1^T} R K^T [K R^T t]_X, \tag{2.10}$$

де для вектора e позначення [e] X обчислюється як

$$[e]_X = \begin{bmatrix} 0 & -e_z & e_y \\ e_z & 0 & -e_x \\ -e_y & e_x & 0 \end{bmatrix}, \tag{2.11}$$

За допомогою фундаментальної матриці обчислюються рівняння епіполярних ліній. Для точки x, вектор, що задає епіполярную лінію, буде мати вигляд l '= F x, а рівняння самої епіполярной лінії: l'T x' = 0. Аналогічно для точки x ', вектор, що задає епіполярную лінію, буде мати вигляд l = FTx'.

Крім фундаментальної матриці, існує ще таке поняття, як істотна матриця (essential matrix): *E = K'TFK*. У випадку, коли матриці внутрішніх

параметрів будуть одиничними істотна матриця буде збігатися з фундаментальної. За суттєвої матриці можна відновити становище і поворот другої камери щодо першої, тому вона використовується в задачах, в яких потрібно визначити рух камери.

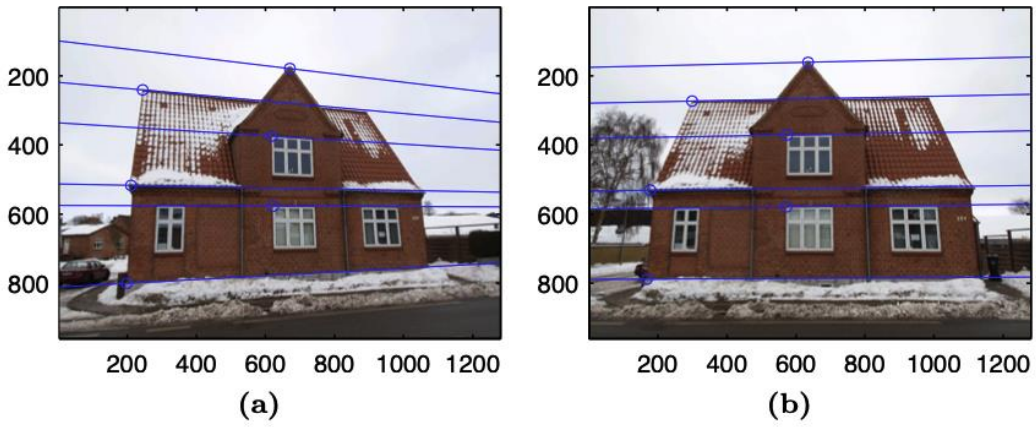


Рисунок 2.3 - Ілюстрація 6 відповідних точок і їх епіполярних ліній на парі зображень. Точка на лівому зображенні визначає на правому зображенні епіполярную лінію, на якій повинна лежати відповідна точка, і навпаки.

Епіполярна геометрія двох камер описується фундаментальної матрицею.

2.3 Особливі точки і дескриптори

Метою кроку порівняння є оцінка відносного положення камер в кожній парі вхідних зображень. Частиною цього процесу є визначення надійного набору відповідних ключових точок на обох зображеннях.

Для встановлення відповідності між ключовими точками на етапі порівняння важливо, щоб ключові точки, виявлені на одному зображенні, були виявлені на іншому зображенні тієї ж сцени; це називається повторюваністю. Крім того, важливо, щоб ключові точки, що походять від зображень одного і того ж об'єкта сцени, могли бути узгоджені; це називається відмітною особливістю. Повторюваність і чіткість ключових точок необхідні навіть при зйомці з різних ракурсів і в різних умовах освітлення. Так само ключові точки не повинні залежати від змін таких параметрів, як масштаб зображення і поворот, зміни точки зору, наявність шуму, наприклад, шуму датчика і стиснення зображення, а також зміни освітленості.

В даний час ведеться велика робота по розробці алгоритмів, спрямованих на надійне виявлення ключових точок, які є інваріантними до цих змін. Два алгоритму, зокрема, добре працюють з широким спектром зображень, а саме алгоритм Scale Invariant Feature Transform (SIFT) [27] і Speed-Up Robust Features (SURF) [14].

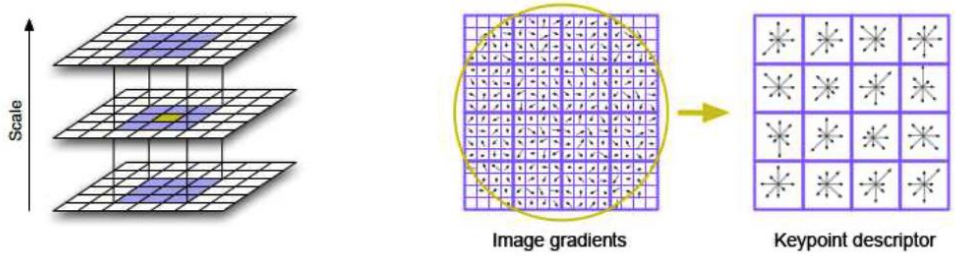
Алгоритми SIFT і SURF схожі, оскільки вони складаються з двох основних етапів: виявлення ключових точок і обчислення їх дескрипторів. Спочатку необхідно знайти всі можливі кандидати ключових точок, а потім обчислити їх дескриптор. Якість дескриптора впливає на стійкість до різних змін, розглянутим вище, і на відмінні риси ключових точок.

SURF є новітнім з двох алгоритмів, і він заснований на SIFT, який дуже добре зарекомендував себе. Були проведені порівняння двох алгоритмів. У роботах [13] та [28] зроблено висновок, що якість виявлених ключових точок

трохи краще для SIFT, але швидкість SURF набагато вище. Однак одна важлива відмінність між алгоритмами очевидно: кількість ключових точок, виявлених на зображенні, значно вище для SIFT. У цьому проекті виявлення великої кількості ключових точок важливіше швидкості, так як в системі немає вимог реального часу, а відновлення структури і руху відбувається краще з великою кількістю ключових точок. Тому в даному проекті для визначення ключових точок вибирається SIFT.

На вході алгоритму SIFT піддається зображення у відтінках сірого, а на виході отримуємо набір ключових точок, виявлених на зображенні. В цілому, два основні кроки алгоритму SIFT виконуються наступним чином. Виявлення точок інтересу здійснюється шляхом побудова однієї піраміди зображень після перетворень маски гаусового розмивання, а друга - пірамід різниці двох сусідніх шарів. На другий шукаються всі локальні екстремуми (рисунок 4.4a). Таким чином, виявлені потенційні точки інтересу незмінні за масштабом і спрямованості [27]. Виділення дескрипторів здійснюється шляхом вимірювання градієнтів локального зображення на виявленому масштабі і перетворення їх в уявлення, що дозволяє отримати значні рівні локальних спотворень форми і змін освітленості (Рисунок 2.4b) [27].

Кожна виявлена ключова точка описується її розташування, масштабом, орієнтацією і дескриптором. На наступних етапах відновлення структури і руху використовуються тільки координати ключової точки в пікселях і дескриптор, що представляє собою 128-мірний вектор.



(a) (b)

Рисунок 2.4 - Два основні кроки алгоритму SIFT. a) Виявлення точок інтересу за допомогою визначення екстремумів в масштабі простору. b) Витяг дескрипторів ключових точок з градієнтів локальних зображень.

2.4 Зіставлення ключових точок

При наявності двох наборів ключових точок, виявлених на одній парі вхідних зображень, метою зіставлення ключових точок є встановлення початкового відповідності між ключовими точками на першому зображенні та ключовими точками на другому зображенні. Для оцінки фундаментальної матриці необхідний початковий набір ключових точок відповідності. Кожна ключова точка описується її розташування, масштабом, орієнтацією і дескриптором.

Кращим кандидатом на ключову точку на першому знімку є найближчий сусід ключовий точки на другому знімку. Найближчим сусідом визначається ключова точка з мінімальним евклідовой відстанню між дескрипторами [27]. Оскільки n і m - це число ключових точок, виявлених на першому і другому зображенні відповідно, нехай di, i =1, ..., n будуть 128 розмірним дескриптором ключовий точки i на першому зображенні. Аналогічно d'j, j = 1, ... m буде дескриптором ключовий точки j на другому зображенні. Тоді найближчим сусідом ключовий точки i на першому зображенні, є точка j на другому зображенні, для якої

$$j = \arg\min_{j} ||\mathbf{d}_i - \mathbf{d}'_j||. \tag{2.12}$$

Однак не всі ключові точки, виявлені на першому знімку, обов'язково відповідають ключовим точкам на другому знімку і навпаки. Тому простого

вибору найближчого сусіда в якості збіги недостатньо, і необхідно знайти спосіб усунути зайві. Метод, запропонований в [27], полягає в порівнянні відстані найближчого сусіда з відстанню другого за величиною сусіда. Нехай d'k буде дескриптором другий за величиною ключовий точки на другому знімку. Тоді дві ключові точки вважаються співпадаючими тільки в тому випадку, якщо вірно наступне співвідношення.

$$\frac{\|\mathbf{d}_i - \mathbf{d}'_j\|}{\|\mathbf{d}_i - \mathbf{d}'_k\|} < \tau \tag{2.13}$$

Значення порога τ, рекомендованого в [27] на основі експериментів, дорівнює 0,8, при цьому 90% помилкових збігів відкидається, а менше 5% правильних збігів - немає. В даному проекті поріг τ = 0,6 використовується за результатами обговорення в наступному розділі.

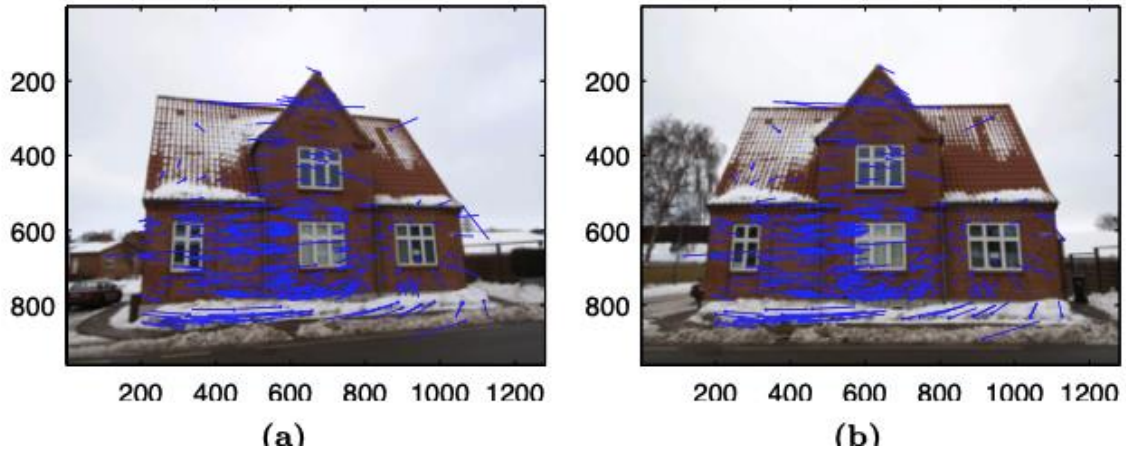


Рисунок 2.5 - Ключова точка збігається між двома зображеннями, показаними в підпунктах (a) і (b), з використанням порогу τ = 0,6. Точки позначають розташування ключових точок на зображенні, а інший кінець ліній вказує розташування співпадаючих ключових точок на іншому зображенні (відображається тільки 25% від 1527 збігів).

Щоб визначити початкове відповідність між ключовими точками на двох зображеннях, можна знайти відповідні пари ключових точок (i, j), застосувавши (2.12) для всіх ключових точок і на першому зображенні, щоб знайти відповідні ключові точки j на другому зображенні, і тільки зберігаючи відповідності для яких (2.13) задоволені. При такому підході, однак, може статися так, що дві або більше ключових точки збігаються з однією ключовою точкою j на другому зображенні. В такому випадку вибирається пара з найменшою відстанню дескрипторів.

2.6 Оцінка фундаментальної матриці

В даному розділі розроблений метод надійної оцінки фундаментальної матриці F і ідентифікації надійного відповідності між ключовими точками на двох знімках, тобто набором ключових точок, які відповідають епіполярному обмеження.

Існують різні алгоритми для оцінки фундаментальної матриці з набору відповідних точок на двох знімках. 7-точковий алгоритм є вирішенням цього завдання, що вимагає мінімальної кількості точок, але цей алгоритм передбачає вирішення нелінійних рівнянь [27]. Більш прямий підхід полягає у використанні 8-точкового алгоритму, який вирішує завдання тільки за допомогою лінійних рівнянь. У деяких випадках цей алгоритм сприйнятливий до шумів, але в [24] було показано, що нормалізована версія цього алгоритму працює дуже добре. Тому в даному проекті для розрахунку фундаментальної матриці використовується нормалізований 8-точковий алгоритм, аналіз якого наведено в наступному розділі.

Результат зіставлення ключових точок може містити викиди, які не задовольняють епіполярному обмеження. Важливо, щоб такі відхилення не враховувалися при розрахунку фундаментальної матриці, так як це призведе до неправильної відносної позиції камер. Тому для досягнення надійності RANdom SAmple Consense (RANSAC) використовується для визначення надійного набору плоскогубців і розрахунку відповідної фундаментальної матриці в тому ж процесі.

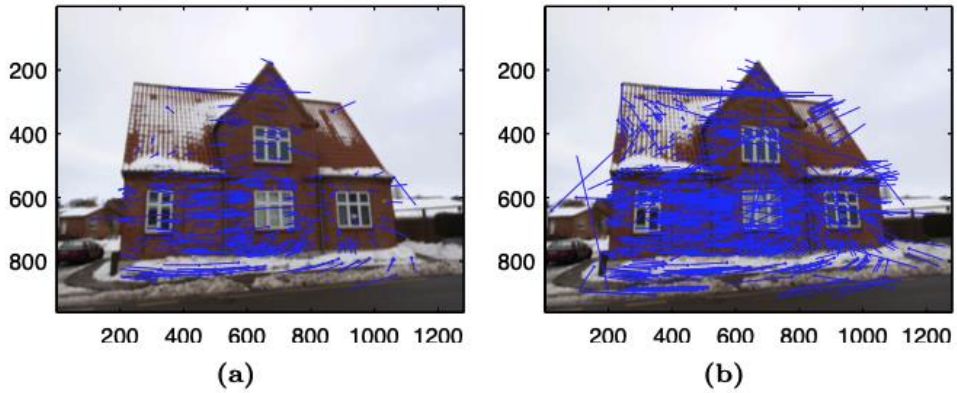


Рисунок 2.6 - Ключова точка збігається між двома зображеннями для різних порогових значень (відображається тільки 25% збігів). Точки позначають розташування ключових точок на зображенні, а інший кінець ліній - розташування збіглися ключових точок на іншому зображенні. a) Поріг τ = 0,6, в результаті 1527 збігів. b) Поріг τ = 0,8, в результаті 3445 збігів. Зверніть увагу на наявність тут декількох помилкових збігів.

2.7 Нормалізований 8-точковий алгоритм

Як випливає з назви, для розрахунку фундаментальної матриці необхідно 8 відповідних точок на двох зображеннях. Дане рішення включає в себе вирішення низки лінійних рівнянь в 8 точках. У разі, якщо відомо більше 8 точок, проблема може бути вирішена за допомогою лінійного мінімізації найменших квадратів. Обчислення фундаментальної матриці за допомогою цього алгоритму просте і з нормалізованої версією можна отримати точні результати [23] і [24].

Лінійне рішення. Оскильки фундаментальна матриця F визначається відношенням x'TFx = 0, де x' і x' є відповідними точками на першому і другому зображенні відповідно. При наявності як мінімум 8 відповідних точок xi і x'i, можна обчислити невідому матрицю F. Тепер даючи x = [xyy 1] T і x '= [x' y '1] T, відповідність кожній точці дає початок одному лінійному

рівнянню в невідомих значеннях F. Це рівняння може бути записано у вигляді координат відомих точок, а саме

$$x'xf11 + x'yf12 + x\ 'f13 + y'x'\ f21 +$$
$$+ y'y\ f22 + y\ 'f23 + xf31 + yf32 + f33 = 0 \qquad (2.14).$$

Додаючи рядок [x'x, x'y, x'y, y'x, y'x, y'y, y'y ', x, y, 1] в матрицю A для кожної точки, набір лінійних рівнянь у вигляді

$$Af = 0 \qquad (2.15)$$

де f - вектор, що містить записи матриці F у великому порядку рядків. Оскільки це однорідний набір рівнянь, вектор f, а значить і F, може бути визначений тільки до невідомого масштабу, і тому додається додаткова умова ||f|| = 1.

Рішення для цієї системи можна знайти за допомогою методу одноразового розкладання значень (SVD), де A розкладається на $A = UDV^T$. Потім f - стовпець V, відповідний найменшому сингулярного значенням A. Факторизація може бути виконана таким чином, що діагональні записи D сортуються в порядку убування, і це передбачається для всіх наступних застосувань SVD. У цьому випадку вектор рішення f є останнім стовпцем V.

Обмежувальне виконання Властивість фундаментальної матриці F, представленої в розділі 5.1, полягає в тому, що вона займає друге місце і, отже, є єдиною. Матриця F, знайдена при вирішенні лінійних рівнянь в (2.15), як правило, не матиме рангу 2 через шум в координатах точок. Для не-єдиною матриці F епіполярние лінії не перетинаються в епіполях двох зображень, що необхідно для отримання достовірної фундаментальної матриці. Тому необхідно забезпечити дотримання обмеження, яке F посідає друге місце.

Це обмеження сингулярності може бути посилено заміною F на матрицю F ', яка зводить до мінімуму норму Frobenius1 || F-F' || = 0. Це може бути досягнуто шляхом повторного використання SVD, а F = UDVT. Якщо діагональна матриця D = di di di (r, s, t) задовольняє r ≥ s ≥ t, то F '= Udiag (r, s, 0) VT - це матриця, яка мінімізує норму Фробениуса ||F-F'|

Нормалізація Вищевказані два кроки, лінійне рішення і застосування обмежень, являють собою базовий 8-точковий алгоритм. Для підвищення точності алгоритму цим крокам може передувати етап нормалізації, за яким слідує етап денормалізації, що і є суттю нормалізованого 8-точкового алгоритму.

Пропонована нормалізація полягає в перетворенні координат точок таким чином, щоб центр ваги точок був на початку координат, а середньоквадратичне відстань точок від початку координат дорівнювало sqrt(2). Перетворення координат точок відповідно до того, де T і T’ нормалізують перетворення, що складаються з перетворення і масштабування. Потім можна знайти фундаментальну матрицю F 'для трансформованих кореспондентів x і x', використовуючи два вищеописаних кроку. Для отримання остаточної оціночної фундаментальної матриці F, що відповідає початкових точок xi і x'i, денормалізуйте, задавши F = T'TF'T.

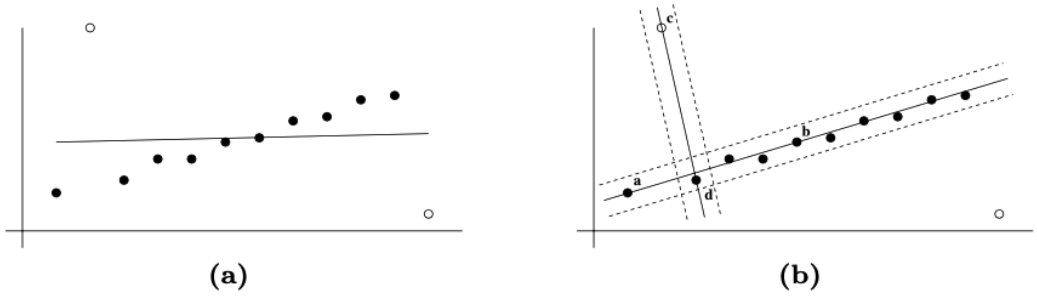


Рисунок 2.7 - Оцінка лінії [23]. Заповнені точки включені, а відкриті точки - викидами. a) Найменша квадратна ортогональна регресія сильно залежить від викидів. b) Дві лінії, які є результатом випадкової вибірки двох

точок в ході RANSAC. Пунктирними лініями позначений поріг відстані. Підтримка для ліній від a до b дорівнює 10, а для ліній від c до d - 2.

2.8 Відновлення відносної позиції камер

З оціночної фундаментальної матриці можна відновити відносне положення обох камер. Сюди входить обчислення матриці, званої головного датчика, про що мова піде в наступному розділі. Потім з основної матриці можна відновити відносну позицію між камерами.

2.8.1 Обчислення базової матриці

Першим кроком відновлення відносної позиції обох камер є обчислення необхідної матриці. Основна матриця E - це матриця 3×3, аналогічна матриці F, але вона може бути застосовна до камер з відомою внутрішнім калібруванням. Тому цей крок залежить від внутрішньої калібрування зображень, отриманих на етапі попередньої обробки.

Матриця камери визначається як P = K [R t], і нехай x = PX буде точкою на зображенні. Знаючи калібровочні матриці камери K, її обернену можна застосувати до точки x для отримання точки $x = K^{-1} x$, яка виражається в нормалізованих координатах. Це усуває вплив внутрішніх параметрів, як показано на сайті x = [R t] X. Матриця камери форми K-1P = [R t] називається нормалізованої матрицею камери [23].

У наступних двох нормалізованих матрицях камери P = [I 0] і P '= [R t], відповідних першому і другому зображенню, відповідно, в парі вхідних зображень. Відомі відповідні калібрувальні матриці камер K і K ', тому для

відповідних точок x і x' можна обчислити нормовані точки x = K-1x і x '= K'-1x'. Основна матриця E тепер визначається наступним чином

$$\hat{\mathbf{x}}'^{\mathsf{T}}\mathbf{E}\hat{\mathbf{x}} = 0. \tag{2.16}$$

Порівняння з (2.1) показує, що основна матриця є фундаментальною матрицею для нормалізованих камер. Можна зробити висновок, що

$$\mathbf{E} = \mathbf{K}'^{\mathsf{T}}\mathbf{F}\mathbf{K}. \tag{2.17}$$

Той факт, що камери нормалізовані накладає додаткові обмеження на основні матриці. Для того щоб матриця була достовірною, вона повинна мати два однакових сингулярних значення, і одне нульове [23]. Для основної матриці E, розрахованої з використанням (2.16), це обмеження може бути застосовано так само, як і обмеження сингулярності для фундаментальної матриці1. Використання SVD дозволяє E = UDVT, D = diag(r, s, t) задовольняє $r \geq s \geq t$. Потім це обмеження може бути посилено обчисленням u = (r + s) / 2 і установкою T кінцевої суттєвої матриці E = Udiag (u, u, 0) V^T.

2.8.2 Відносна позиція основної матриці

З основної матриці E можна відновити відносну позицію між двома камерами до невідомого масштабу. При використанні нормалізованих матриць камер P = [I 0] і P '= [R t] відносна позиція визначається матрицею обертання R і вектором трансляції t. Наступне засноване на [23] і [24].

Незалежно від невідомого масштабу, SVD суттєвої матриці можна записати як E = Udiag (1,1,0) V^T, при цьому U і V вибирають таким чином, що det(U)> 0 і det(V)> 0. Потім вектор перекладу t дорівнює + u3 або -u3, а значить || t | = 1. представляючи матрицю

$$W = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.18}$$

матриця обертання R дорівнює або $R_a = UWV^T$, або $R_b = UW^TV^T$. Це призводить до 4 можливих варіантів вирішення для P ', запропонованим

$$\begin{aligned} P'_A &= [R_a \ \mathbf{u}_3], & P'_B &= [R_a \ -\mathbf{u}_3], \\ P'_C &= [R_b \ \mathbf{u}_3], \text{ and} & P'_D &= [R_b \ -\mathbf{u}_3]. \end{aligned} \tag{2.19}$$

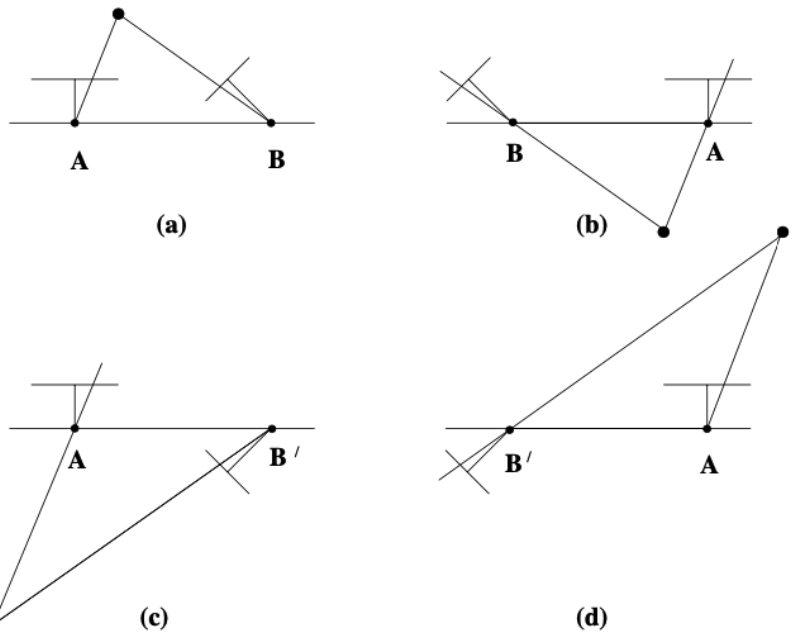


Рисунок 2.8 - Геометрична ілюстрація чотирьох можливих конфігурацій камер [23]. Між лівою і правою точками відліку відбувається

зворотний зсув. Між верхнім і нижнім рядами камера B повертає камеру 180◦ щодо базової лінії. Тільки в конфігурації (a) перед обома камерами знаходиться тріангулірованная точка.

Тільки одна з них відповідає істинній конфігурації, і щоб визначити, яка з них, накладається обмеження ширини. Шляхом тріангуляції3 просторової точки X з двох відповідних точок на зображеннях за допомогою матриць камери P і P'A можна усунути двозначність. Чотири рішення показані на малюнку 2.8. Як видно з малюнка, точка тріангуляції знаходиться тільки перед обома камерами в одній з конфігурацій.

Нехай X = [X1,X2,X3,X4] T буде однорідним 4-вектором, що представляє тріангулірованную точку, а c1 = X3X4 і c2 = (P'AX3)X4. Потім, якщо c1> 0, точка знаходиться перед першою камерою, а c2> 0 - перед другою камерою. Отже, якщо c1> 0 і c2> 0, справжня конфігурація буде P' = P'A. Якщо c1 <0 і c2 <0, то базова лінія буде зворотним, а P '= P'B. З іншого боку, якщо c1c2 <0 необхідно перевірити виту пару, то обчисліть c3 = (P'C X) 3 X4. Якщо c1> 0 і c3> 0, то істинної конфігурацією буде P '= P'C. Нарешті, якщо c1 <0 і c3 <0 базова лінія розгортається і P '= P'D

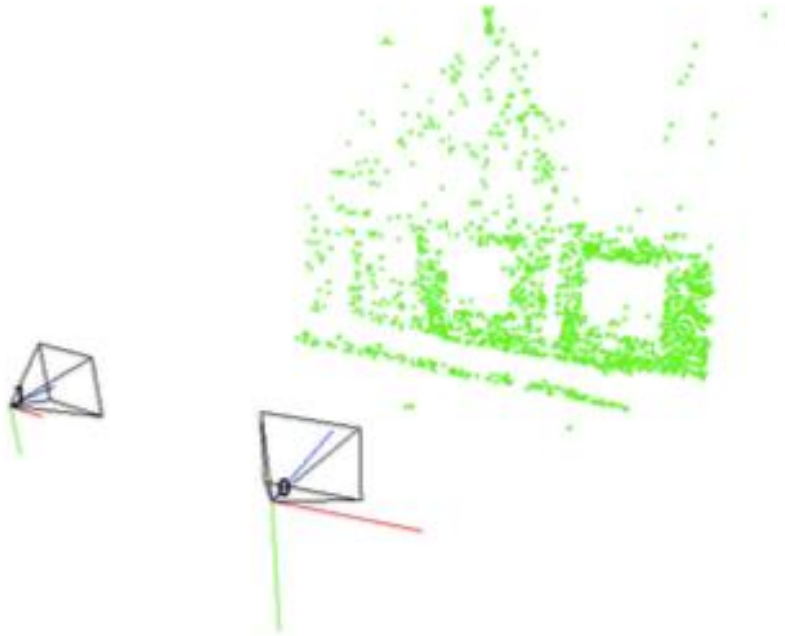


Рисунок 2.9 - Приклад відносної позиції, відновленої для пари вхідних зображень. На додаток до камер з тріангуляцією ключових точок показані зеленими точками.

2.9 Реконструкція великогабаритної моделі

У цьому розділі описаний етап реконструкції великогабаритної моделі запропонованого методу реконструкції. Це єдиний крок, який вимагає взаємодії з користувачем, і мета полягає в отриманні великогабаритної моделі будівлі для реконструкції. Велика модель являє собою текстуровану сітку з багатокутниками, що представляє собою великі плоскі поверхні будівлі, такі як стіни, покрівля і т.д. Користувач несе відповідальність тільки за визначення форми великогабаритної моделі. Зовнішній вигляд моделі, тобто текстури окремих полігонів, автоматично витягується з вхідних зображень.

2.10 Відновлення за участі користувача

Реконструкція великогабаритної моделі виконується інтерактивно за допомогою програми з простим призначеним для користувача інтерфейсом, розробленого спеціально для цієї мети.

Простий підхід до реконструкції великогабаритних моделей був обраний, так як основним фокусом проекту по реконструкції моделей є автоматична реконструкція фасадів. Метод реконструкції моделі за допомогою користувача, реалізований в даному проекті, складається з двох етапів. Першим кроком є визначення набору локаторів, які представляють собою 3D-позиції обраних об'єктів будівлі, наприклад, кутів стіни. Другим кроком є визначення полігонів, що покривають плоскі поверхні будівлі, з використанням певних локаторів як вершин полігонів. Два кроки можуть виконуватися в будь-якому порядку, поки перед самим полігоном не будуть визначені всі локатори, необхідні для конкретного полігона.

В інтерфейсі можна переключатися між вхідними зображеннями кластера, а локатор визначається мишею простим клацанням миші по одній і тій же деталі будівлі в двох або більше вхідних зображеннях. Цей процес показаний на малюнку 2.10.

Потім 3D-положення локатора оцінюється за допомогою параметрів калібрування камери, доступних на етапах попередньої обробки і кластеризації. Зокрема, вкажіть за допомогою li, j заданий користувачем положення в піксельних координатах локатора i на знімку j. При наявності двох таких точок li, a і li, b, відповідають одному і тому ж ознакою на зображеннях a і b відповідно, можна отримати оцінку 3D положення локатора Li за допомогою методу тріангуляції. Для тріангуляції використовуються матриці камер Pa і Pb і дві точки зображення, виражені в нормалізованих координатах. Нормалізовані координати виходять шляхом

застосування до піксельним координатам зворотного калібрувальної матриці камери K. Для простоти, місце розташування локатора j оцінюється з використанням тільки двох з певних користувачем розташування.

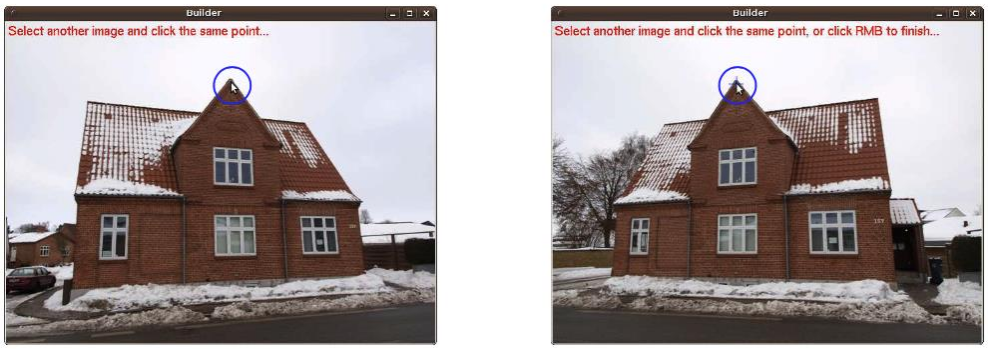


Рисунок 2.10: Інтерактивне визначення локатора. a) Користувач натиснув на елемент на одному зображенні, виділеному синім кружком. b) Користувач переключився на інше зображення і натиснув на той же елемент. З заданих розташування новий локатор тріангулірується.

В інтерфейсі полігон визначається простим натисканням на локатори для використання в якості вершин в порядку проти годинникової стрілки, коли полігон видно спереду, що показано на рисунку 2.11. Кожен полігон грубої моделі описаний послідовністю отриманих таким чином індексів локаторів.

При визначенні полігону, частково побудований полігон відображається інтерактивно, а приблизна текстура витягується в режимі реального часу, таким чином, користувач отримує негайну візуальну зворотний зв'язок. Після того, як користувач закінчив визначення полігону грубої моделі, для цього полігону витягується вишукана текстура.

Як вже говорилося вище, користувач відповідає тільки за визначення форми грубої моделі, тобто вибір об'єктів на вхідних зображеннях для визначення локаторів і полігонів на основі цих локаторів. На Рисунку 7.11

показана геометрія відновленої за допомогою цього методу великогабаритної моделі та структури.

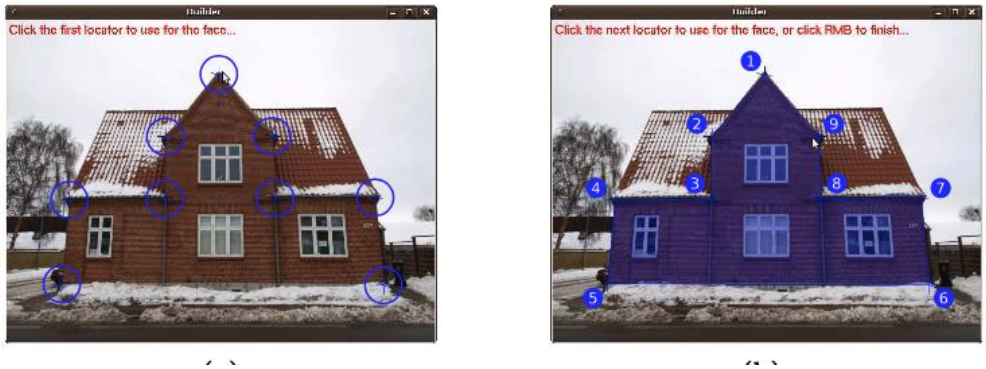


Рисунок 2.11 - Інтерактивне визначення полігону. a) Користувач почав створення полігону після визначення 9 локаторів, виділених синіми колами. b) Локатори натискаються в зазначеному порядку, і створений полігон накладається на зображення.

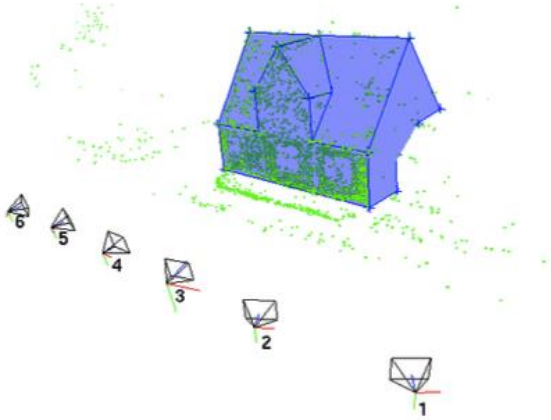


Рисунок 2.12: Приклад геометрії реконструйованої великогабаритної моделі. На додаток до моделі показана частина відновленої структури і руху.

При цьому не можна примусити до того, щоб вершини полігону в грубій моделі, тобто положення підмножини певних локаторів, лежали в одній і тій же площині. Просто передбачається, що вершини певних полігонів близькі до копланарним. Однак те, наскільки це припущення вірне, багато в чому залежить від точності визначення користувачем місця

розташування. Для великих моделей, реконструйованих на основі наборів даних, використаних в даному проекті, вплив цього допущення на результати несуттєво.

Як згадувалося вище, для простоти визначення місця розташування локаторів використовуються два певних користувачем місця розташування. Більш високу точність можна отримати, якщо використовувати більш двох точок зображення, якщо вони надані користувачем. Один із способів оптимізації положення локатора полягає в використанні тріангуляції для отримання первісної оцінки, а потім застосуванні регулювання пучка, при якій параметри камери залишаються незмінними. Однак точність, отримана простим методом, реалізованим в даному проекті, достатня для використання результатів при розробці методу автоматичної реконструкції фасаду.

Одна з ідей для вилучення текстури полягає в використанні вхідного зображення, яке найкраще відображає полігон безпосередньо як текстуру для цього полігону. На жаль, візуальне якість реконструйованої моделі, отримане при такому підході, незадовільно. Проблема полягає в тому, що зображення поверхні будівлі на вхідному зображенні в переважній більшості випадків піддається перспективному перетворенню. Тобто поверхню будівлі не лежить в площині, паралельній площині вхідного зображення, і тому виглядає спотвореною на зображенні.

Локатори, які використовуються для визначення вершин полігону, можуть перебувати не в одній і тій же площині, оскільки їх положення оцінюється по точкам зображення, заданим користувачем. Тому першим кроком в отриманні текстури є пошук площині, яка найкраще підходить для локаторів, що визначають полігон. Остаточна вирівняна текстура буде містити зовнішній вигляд поверхні будівлі, як якщо б зображення було отримано шляхом наведення камери безпосередньо на поверхню під прямим кутом. Тобто, площину зображення цієї віртуальної камери буде паралельна площині, знайденої на цьому етапі.

Далі припустимо, що один полігон моделі визначається послідовністю індексів локатора від 1 до n. Тоді вершинними положеннями цього полігону є оціночні положення локатора L_i, i=1,....n. Підгонка площині тепер є проблемою знаходження площині, яка мінімізує ортогональні відстані від цих локаторів до площини. Центроид m локаторів лежить в цій площині і може бути обчислений наступним чином

$$\mathbf{m} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{L}_i.$$

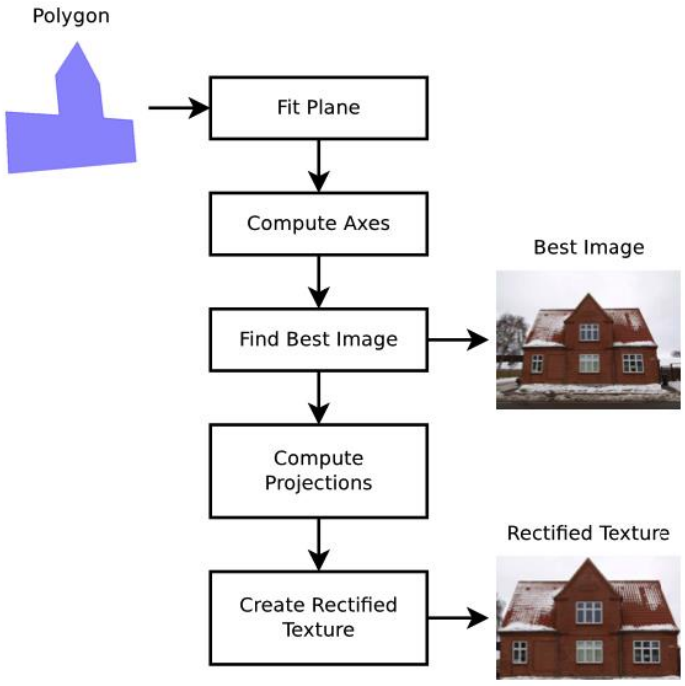


Рисунок 2.13 - Огляд кроків, пов'язаних з отриманням виправленої текстури для полігону грубої моделі.

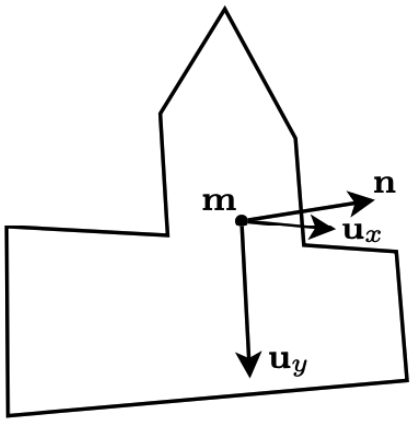
Позначаючи n одиницею виміру нормальний вектор встановленої площині, ортогональное відстань від локатора L_i до площини стає $(L_i\text{-}m)n$. Встановлена площину визначається відомим центром тяжіння m і нормальним n, який повинен бути знайдений. Шукати рішення найменших квадратів, і ця задача може бути виражена як знаходження нормального n, яке мінімізує суму

$$\sum_{i=1}^{n} \left((\mathbf{L}_i - \mathbf{m}) \cdot \mathbf{n} \right)^2.$$

Найкраща підходяща площину повністю описується центроїдом m і одиницею нормальної n. Однак для створення випрямленою текстури необхідно додатково визначити дві осі, перпендикулярні один одному і лежать в площині. Ці осі визначають осі X і Y випрямленою текстури. Для осей існує нескінченна кількість варіантів вибору, так як вони можуть вільно обертатися навколо нормальної площини, і тому необхідно вибрати певну орієнтацію.

У цьому проекті вісь x просто вибирається таким чином, щоб вона була паралельна самому довгому краю полігону. Наприклад, найдовшим краєм полігону є край між локаторами 5 і 6, і вісь X цього полігону стає паралельної цього краю. Однак такий підхід може виявитися не оптимальним у всіх ситуаціях.

Точніше, вкажіть ux і uy векторами перпендикулярних одиниць, що визначають вісь x і y в просторі площини відповідно. Перший ux обчислюється як одиничний вектор, який має той же напрямок, що і найдовший край полігону, що проектується на площину. Потім вісь Y обчислюється як $u_y=u_x \times n$, і вектори ux, uy і n утворюють ортонормального основу. Приклад площинних осей, отриманих для полігону, показаний на Рисунку 2.14.



Рисунку 2.14 - Приклад осей u_x і u_y, розрахованих для площині, яка визначається центроїдом m і нормальної n. Ось x площині має такий же напрямок, що і найдовший край полігону.

Для створення випрямленою текстури полігону необхідно визначити, яке з вхідних зображень найкраще підходить для перегляду полігону. Насправді може не бути жодного зображення, що містить незайняте зображення всього полігону, тому може виникнути необхідність об'єднати кілька вхідних зображень, щоб уникнути цієї проблеми. В даному проекті, проте, передбачається, що для текстури полігону досить одного вхідного зображення, і тому цей крок зводиться до вибору найкращого з доступних зображень.

Один з підходів до вирішення цієї проблеми полягає у виборі зображення, яке найбільш безпосередньо переглядає весь полігон. Тобто, вибираючи вхідне зображення лінія зору якої має найменший кут щодо нормального вектора площини багатокутника.

У наступних розділах давайте позначимо індекс зображення, яке найкраще підходить для перегляду полігону. Потім за допомогою методу, розробленого вище, це зображення виявляється як зображення, для якого досягає максимального значення не менше τ, з додатковим обмеженням, що все локатори видно на зображенні. Якщо зображення, яке задовольняє цим

вимогам, відсутня в кластері, виправлена текстура не може бути залучена з полігону, а замість неї використовується фіктивна текстура.

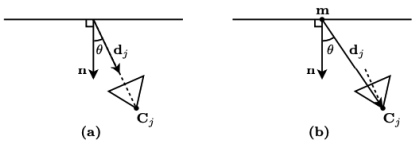


Рисунок 2.15 - Ілюстрація двох підходів до вибору найкращого зображення. a) Використовуйте кут між площиною нормальної і лінією огляду камери. b) Використовуйте кут між площиною нормальної і вектором від центра ваги до камери.

РОЗДІЛ 3 РЕЗУЛЬТАТ РОБОТИ ПРОГРАМНОГО ПРОДУКТУ

Всього для оцінки системи було використано 11 наборів даних. Кожен з наборів даних складається з невпорядкованого набору зображень будівлі, отриманих з різних точок зору. На Рисунку 3.1 наведено приклади всіх зображення з набору даних hassundvej-a. Кількість зображень в наборах даних варіюється від 6 до 10.

Всі зображення в наборах даних були зменшені до 1280 × 960 пікселів. Цей дозвіл було вибрано як баланс між візуальним якістю реконструюються моделей, обчислювальної складністю, кількістю ключових точок. Зображення містять дані EXIF, на камері разом із зображенням при зйомці.

У таблиці 10.1 наведені результати етапу попередньої обробки. У таблиці вказано кількість зображень в кожному з наборів даних і середня кількість ключових точок, виявлених на знімках.

Результати етапу узгодження наведені в таблиці 10.2. Загальна кількість пар зображень в наборі даних одно біноміальному коефіцієнту m2, де m - кількість зображень в наборі даних. Для того щоб пара зображень вважалася прийнятною, кількість ключових точок повинна складати не менше 100.

Нарешті, в таблиці 3.3 перераховані результати етапу кластеризації для кожного з наборів даних, а на малюнку 4.1 показана відновлена структури для чотирьох наборів даних.

З Таблиці 3.1 видно, що в цілому на вхідних зображеннях виявлено велику кількість ключових точок. Кількість виявлених ключових точок залежить від вмісту зображення. Однак, як видно з таблиці 10.2, середня кількість збігів ключових точок між знімками значно менше, ніж кількість ключових точок на кожному зі знімків. Виявлені ключові точки менш різняться з-за повторюваних закономірностей, і тому для порівняння ключових точок був використаний низький поріг. Порівняння числа збігів і

відхилень ключових точок в таблиці 10.2 показує, що використовуваний поріг призводить до невеликих відхилень.

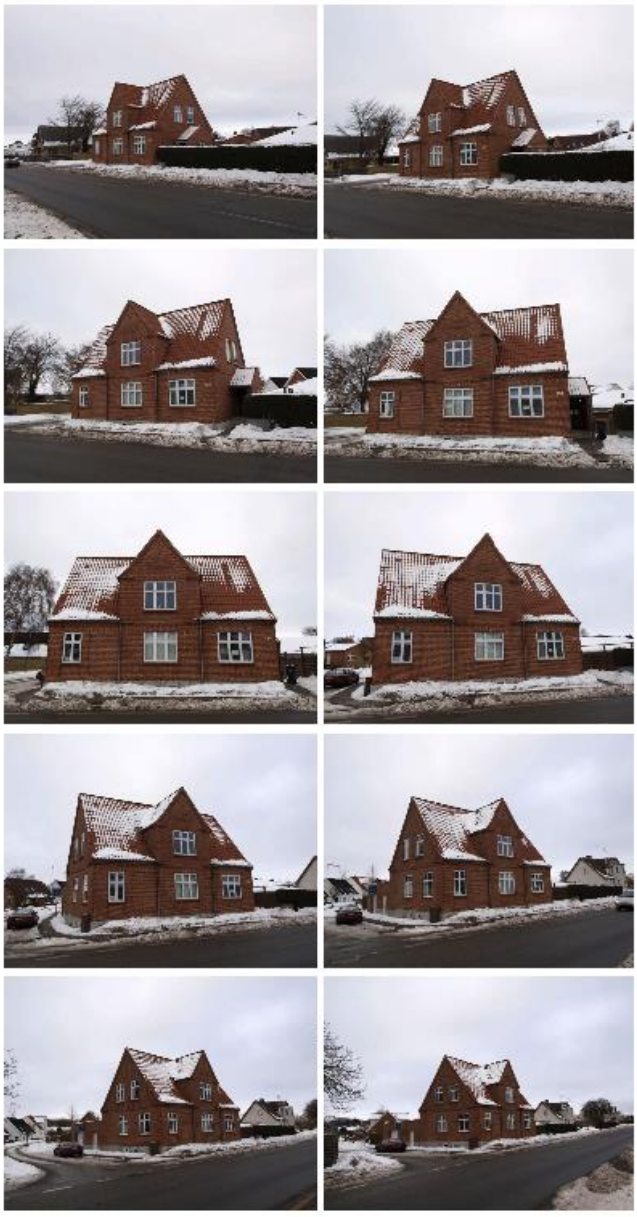


Рисунок 3.1 - Всі 10 зображень в наборі даних building-a.

Таблиця 3.1 - Результати препроцессірованія. У стовпці Keypoints зазначено середня кількість ключових точок на одне зображення.

Датасет	**Зображень**	**Ключових точок**
bernstorffsgade	10	6,204
bjoernoegade	7	5,839
hadsundvej-a	10	8,589
hadsundvej-b	8	4,758
hadsundvej-c	7	4,597
hadsundvej-d	7	4,398
oestrealle-a	9	7,300
oestrealle-b	6	9,419
oestrealle-c	6	6,333
riishoejsvej-a	8	5,518
riishoejsvej-b	7	7,827

Таблиця 3.2 - Результати кластеризації. У стовпці "Проекцій" в дужках вказано середню кількість проекцій на точку.

Датасет	**Камер**	**Точок**	**Проекцій**	**Помилка**
bernstorffsgade	10	4,087	9,790 (2.4)	0.20
bjoernoegade	7	875	1,786 (2.0)	0.22
hadsundvej-a	10	4,426	10,196 (2.3)	0.22
hadsundvej-b	8	821	1,752 (2.1)	0.32
hadsundvej-c	6	755	1,561 (2.1)	0.27
hadsundvej-d	7	1,690	4,103 (2.4)	0.30
oestrealle-a	9	2,238	4,989 (2.2)	0.31
oestrealle-b	6	1,091	2,279 (2.1)	0.21
oestrealle-c	6	1,397	2,993 (2.1)	0.22
riishoejsvej-a	7	870	2,180 (2.5)	0.29
riishoejsvej-b	7	3,754	12,214 (3.3)	0.22

Таблиця 3.3 - Результати зіставлення.

Датасет	**Пари ОТ**	**Особливих точок**
bernstorffsgade	20 (45)	440
bjoernoegade	10 (21)	285
hadsundvej-a	14 (45)	557
hadsundvej-b	7 (28)	200
hadsundvej-c	5 (21)	240
hadsundvej-d	12 (21)	312
oestrealle-a	13 (36)	340
oestrealle-b	6 (15)	267
oestrealle-c	5 (15)	374
riishoejsvej-a	9 (28)	189
riishoejsvej-b	21 (21)	960

Таблиця 3.4 - Статистика для реконструйованих великогабаритних моделей.

Датасет	**Локаторів**	**Полігонів**
bernstorffsgade	22	11
bjoernoegade	8	3
hadsundvej-a	20	9
hadsundvej-b	20	13
hadsundvej-c	15	7
hadsundvej-d	7	2
oestrealle-a	33	21
oestrealle-b	8	3
oestrealle-c	11	4
riishoejsvej-a	9	2
riishoejsvej-b	10	4

Рисунок 3.2 - Структура з результатів створення мапи ключових точок для наборів даних building-b, building-c. a) Два вхідних зображення. b) Відновлена структура.

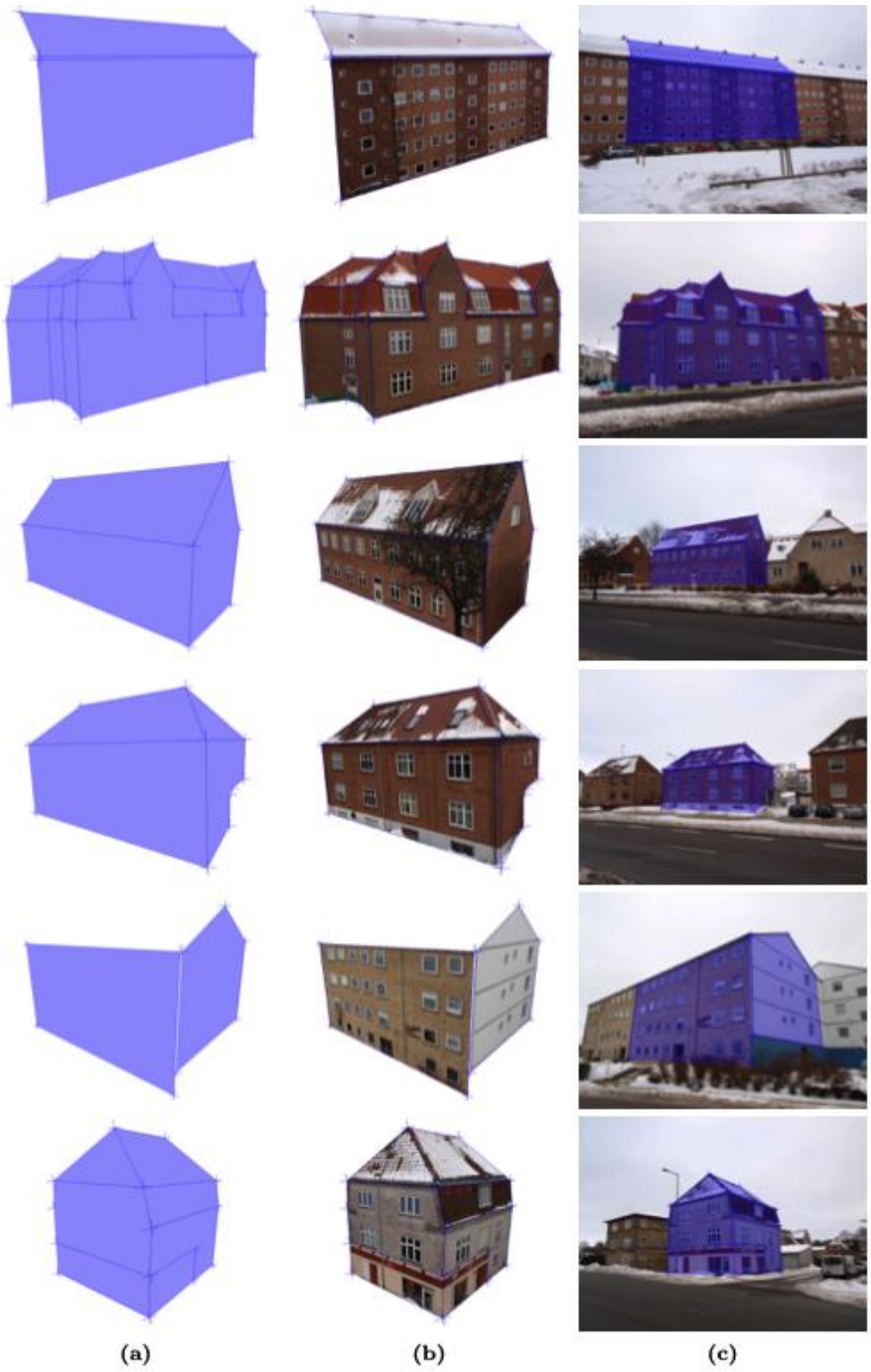


Рисунок 3.3 - Результати реконструкції великогабаритних моделей для наборів даних bjoernoegade, oestrealle-a, oestrealle-b, oestrealle-c, riishoejsvej-a, і hassundvej-c (зверху). a) Геометрична модель. b) Текстурована модель. c) Геометрична модель, накладена на зображення.

РОЗДІЛ 4 РОЗРОБКА СТАРТАП-ПРОЕКТУ

4.1 Опис ідеї та технологічний аудит стартап-проекту

В межах даного підрозділу послідовно проаналізовано та подано у вигляді таблиць наступні пункти:

- зміст ідеї;
- можливі напрямки застосування;
- основні вигоди, що може отри мати користувач товару (за кожним напрямом застосування);
- чим відрізняється від існуючих аналогів та замінників;

Перші три пункти подано у вигляді таблиці (Таблиця 4.1) і дають уявлення про зміст ідеї та потенційні ринки, в межах яких потрібно шукати групи потенційних клієнтів.

Таблиця 4.1 – Опис ідеї стартап проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Надання модуля та застосунку для реконструкції 3Д моделі обєкту з набору фотозображень	Контроль роботи співробітників у системі	Контроль часу входу/виходу із системи зі сторони керівництва
	Використання модуля для створення додатків віртуальної реальності	Зручна та проста інтеграція модуля та пришвидшення створення власних додатків

Аналіз потенційних техніко-економічних переваг ідеї порівняно із пропозиціями конкурентів передбачає:

- визначення переліку техніко-економічних властивостей та характеристик ідеї;
- визначення попереднього кола конкурентів, проектів-конкурентів, товарів-замінників чи товарів-аналогів, що вже існують на ринку;
- збір інформації щодо значень техніко-економічних показників для ідеї власного проекту та проектів-конкурентів.

Відповідно до визначеного вище переліку проводиться порівняльний аналіз показників: гірші значення (W, слабкі); аналогічні (N, нейтральні) значення; кращі значення (S, сильні).

Визначення сильних, слабких та нейтральних характеристик ідеї стартап-проекту “ Модуль тривимірної реконструкції моделей об'єкта з послідовності фотозображень ” наведено у Таблиці 4.2.

Таблиця 4.2 – Визначення сильних, слабких та нейтральних характеристик

№	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів			Х-ка
		Мій проект	Конк. 1	Конк 2	
1	Форма виконання	Десктопний застосунок	Консольний застосунок	Web застосунок	S
2	Собівартість	Низька	Низька	Низька	N
3	Точність результатів	Середня	Велика	Низька	N
4	Наявність інтернету	Ні	Ні	Так	W
5	Кросплатформеність	Так	Так	Так	N
6	Складність використання/ автономність	Ні	Так	Так	S

Визначений перелік слабких, сильних та нейтральних характеристик та властивостей ідеї потенційного товару є підґрунтям для формування його конкурентоспроможності.

4.2 Технологічний аудит ідеї стартап-проекту

В межах даного підрозділу необхідно провести аудит технології, за допомогою якої можливо реалізувати ідею проекту (технології створення товару).

Визначення технологічної здійсненності ідеї проекту передбачає аналіз таких складових:

- за якою технологією буде виготовлено товар згідно ідеї проекту;
- чи існують такі технології, чи їх потрібно розробити/доробити;
- чи доступні такі технології авторам проекту?

Технологічну здійсненність ідеї стартап-проекту “ Модуль тривимірної реконструкції моделей об'єкта з послідовності фотозображень ” наведено у Таблиці 4.3.

Таблиця 4.3 – Технологічна здійсненність ідеї стартап-проекту

№	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Модуль тривимірної реконструкції моделей об'єкта з послідовності фотозображень	C#	Наявна	Безкоштовна, доступна
		Java	Наявна	Безкоштовна, доступна
		Python	Наявна	Безкоштовна, доступна

Обрана технологія реалізації ідеї проекту для створення застосунку для виявлення аномальної роботи операційної системи Windows обрана технологія C#, яка є безкоштовною та якою володіють розробники.

4.3 Аналіз ринкових можливостей запуску стартап-проекту

Визначення ринкових можливостей, які можна використати під час ринкового впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту, дозволяє спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів. Спочатку проводиться аналіз попиту: наявність попиту, обсяг, динаміка розвитку ринку (Таблиця 4.4).

В ході таких досліджень вивчаються особливості і перспективи розвитку попиту на конкретні товари, позиції конкурентів на ринку, їх сильні і слабкі сторони, динаміку цін і т.д. Стартап-проекту важливо знати, чи буде обсяг продажів його товарів достатнім для компенсації зусиль щодо виходу на ринок, тому важливою характеристикою ринку є його ємність, під якою розуміють максимально можливий обсяг продажу певного товару протягом року, виражений в натуральних і вартісних одиницях.

Попит на більшість товару, який визначає місткість ринку, характеризується нестабільністю. Тому кожне підприємство прагне мати достовірний прогноз попиту на свій товар. З метою стимулювання збільшення попиту на товар необхідно вивчити і проаналізувати думки і потреби споживачів певного товару.

Попередню характеристику потенційного ринку стартап-проекту “Модуль тривимірної реконструкції моделей об'єкта з послідовності фотозображень” наведено у Таблиці 4.4.

Таблиця 4.4 – Попередня характеристика потенційного ринку стартап-проекту

Показники стану ринку (найменування)	Характеристика
Кількість головних гравців, од	2
Загальний обсяг продаж, грн/ум.од	180000
Динаміка ринку (якісна оцінка)	Зростає
Наявність обмежень для входу (вказати характер обмежень)	Немає
Специфічні вимоги до стандартизації та сертифікації	Немає
Середня норма рентабельності в галузі (або по ринку), %	R=22%

Так як застосунок має безпосереднє відношення до роботи операційної системи та її безпечної та стабільної роботи, можна зробити висновок, що нашим сегментом ринку компанії будуть:

Таблиця 4.5 – Потенційні сегменти споживачів

Характеристика	Сегмент 1	Сегмент 2	Сегмент 3	Сегмент 4
Тип компанії	product	outsource	product	outsource
Розмір	800+	800+	21-80	21-80
Сфера	IT			
Ємність	6 000	4 000	20 000	120 000

Характеристику потенційних клієнтів стартап-проекту “ Модуль тривимірної реконструкції моделей об'єкта з послідовності фотозображень ” наведено у Таблиці 4.6.

Таблиця 4.6. – Характеристика потенційних клієнтів стартап-проекту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Витрачати менше часу на аналіз коректності роботи операційної системи	Аудиторія: системні адміністратори, розробники. Сегменти: індивідуальні користувачі, маленькі підприємства, великі підприємства.	Для сегменту маленьких підприємств та індивідуальних користувачів характерне разове використання застосунку, в той час як великі компанії можуть поставити його використання на автоматичний режим	Мати розуміння роботи операційної системи, мати досвід роботи з подібними застосунками.

Після визначення потенційних груп клієнтів проводиться аналіз ринкового середовища: складаються таблиці факторів, що сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають. Фактори в таблицях подають в порядку зменшення значущості.

Ринкові можливості – це сприятливі обставини, які підприємство може використовувати для отримання переваг. Слід зазначити, що можливостями з погляду SWOT-аналізу є не всі можливості, які існують на ринку, а тільки ті, які можна використовувати.

Ринкові загрози – події, настання яких може несприятливо вплинути на підприємство.

Фактори загроз стартап-проекту “Модуль тривимірної реконструкції моделей об'єкта з послідовності фотозображень” наведено у Таблиці 4.7. Фактори можливостей стартап-проекту Модуль тривимірної реконструкції моделей об'єкта з послідовності фотозображень ” наведено у Таблиці 4.8.

Таблиця 4.7. - Фактори загроз

№ п/п	**Фактор**	**Зміст загрози**	**Можлива реакція компанії**
1	Зростаюча конкуренція	Зі зростанням попиту на розробку додатків для аналізу ризиків зросла і пропозиція.	Розробляти додаток високої якості та з додатковими унікальними функціями.
2	Зміна потреб користувачів	Користувачам необхідне програмне забезпечення з іншим функціоналом	Передбачити можливість додавання нового функціоналу до створюваного ПЗ

Таблиця 4.8 – Фактори можливостей

№ п/п	**Фактор**	**Зміст можливості**	**Можлива реакція компанії**
1	Зростаючий попит	Збільшення попиту на додаток.	Надавати високоякісні рішення, займати нішу ринку.
2	Оптимізація швидкості	Оптимізація швидкості завантаження додатка.	Оптимізація швидкості завантаження за рахунок

	завантаження		рефакторингу, асинхронності, мінімізації файлів кінцевого веб-застосування та оптимізації стиснення зображень.
3	Зниження довіри до конкурента 1	У конкурента №1 нещодавно була знайдена помилка, завдяки якій дані клієнтів стали доступні в інтернеті	При виході на ринок звертати увагу покупців на безпеку нашого ПЗ та авторитетність компанії

Ступеневий аналіз конкуренції на ринку стартап-проекту “Модуль тривимірної реконструкції моделей об'єкта з послідовності фотозображень” наведено у Таблиці 4.9.

Таблиця 4.9 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	**В чому проявляється дана характеристика**	**Вплив на діяльність підприємства**
1. Вказати тип конкуренції - досконала	Існує 2 компанії-конкуренти на ринку	Врахувати ціни конкурентних компаній на початкових етапах створення бізнесу, реклама (вказати на конкретні переваги перед конкурентами)
2. За рівнем конкурентної боротьби - міжнародний	Всі компанії з інших країн	Використовувати локалізацію
3. За галузевою ознакою - внутрішньогалузева	Конкуренти мають ПЗ, яке використовується лише всередині даної галузі	Створити основу ПЗ таким чином, щоб можна було легко переробити дане ПЗ для використання у інших галузях та додавати нові модулі в існуюче
4. Конкуренція за видами товарів: - товарно-видова	Види товарів є однаковими	Створити ПЗ, враховуючи недоліки конкурентів
5. За характером конкурентних переваг - нецінова	Вдосконалення технології створення ПЗ, щоб собівартість була нижчою	Використання менш дорогих технологій для розробки, ніж використовують конкуренти
6. За інтенсивністю - марочна	Бренди присутні	-

Після аналізу конкуренції проведено більш детальний аналіз умов конкуренції в галузі (табл. 4.10).

Таблиця 4.10 - Аналіз конкуренції в галузі за М. Портером

Складові аналізу		**Висновки**
Прямі конкуренти в галузі	Навести перелік прямих конкурентів	Існує 2 конкуренти на ринку. Найбільш схожим за виконанням є конкурент 1, так як його рішення має велику точність.
Потенційні конкуренти	Визначити бар'єри входження в ринок	Так, можливості для входу на ринок є, бо наше рішення покращує та пришвидшує роботу спеціаліста
Постачальники	Визначити фактори сили постачальників	Постачальники відсутні.
Клієнти	Визначити фактори сили споживачів	Важливим для користувача є кросплатформеність ПЗ та якість його роботи.
Товари-замінники	Фактори загроз з боку замінників	Товари-замінники можуть використати більш дешеву технологію створення ПЗ та зменшити собівартість товару.

За результатами аналізу таблиці зроблено висновок щодо принципової можливості роботи на ринку з огляду на конкурентну ситуацію. Також зроблено висновок щодо характеристик (сильних сторін), які повинен мати проект, щоб бути конкурентоспроможним на ринку. Другий висновок враховується при формулюванні переліку факторів конкурентоспроможності. На основі аналізу конкуренції, проведеного в табл.4.10, а також із

урахуванням характеристик ідеї проекту (табл. 4.2), вимог споживачів до товару (табл. 4.6) та факторів маркетингового середовища (табл. 4.7 – табл. 4.8) визначено та обґрунтовано перелік факторів конкурентоспроможності. Аналіз оформляється за табл. 4.11.

Таблиця 4.11 - Обґрунтування факторів конкурентоспроможності

№	**Фактор конкуренто-спроможності**	**Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)**
1	Виконання ПЗ у вигляді зручного у користуванні десктопного застосунку	Це рішення дозволяє швидко встановлювати використовувати ПЗ на комп’ютер користувача
2	Простота інтерфейсу користувача	Інтерфейс користувача зроблений таким чином, що користувачу необхідно лише заповнити необхідні поля.
3	Наявність моделей ШІ	Це дозволить надати користувачеві інформацію, яка може спростити його работу

За визначеними факторами конкурентоспроможності (табл. 4.11) проведено аналіз сильних та слабких сторін стартап-проекту (табл. 4.12).

Таблиця 4.12 - Порівняльний аналіз сильних та слабких сторін проекту

№	**Фактор конкуренто-спроможності**	**Бали 1-20**	**Рейтинг товарів-конкурентів у порівнянні з нашим підприємством**						
			-3	-2	-1	0	1	2	3

1	Десктопний застосунок	19			+				
2	Простота інтерфейсу користувача	16	+						

Фінальним етапом ринкового аналізу можливостей впровадження проекту є складання SWOT-аналізу (матриці аналізу сильних (Strength) та слабких (Weak) сторін, загроз (Troubles) та можливостей (Opportunities) (табл. 4.13) на основі виділених ринкових загроз та можливостей, та сильних і слабких сторін (табл. 4.12). Перелік ринкових загроз та ринкових можливостей складено на основі аналізу факторів загроз та факторів можливостей маркетингового середовища. Ринкові загрози та ринкові можливості є наслідками (прогнозованими результатами) впливу факторів, і, на відміну від них, ще не є реалізованими на ринку та мають певну ймовірність здійснення. Наприклад: зниження доходів потенційних споживачів – фактор загрози, на основі якого можна зробити прогноз щодо посилення значущості цінового фактору при виборі товару та відповідно, – цінової конкуренції (а це вже – ринкова загроза).

Таблиця 4.13 - SWOT- аналіз стартап-проекту

Сильні сторони: простий інтерфейс користувача, кросплатформенність ,достатня точність результатів	**Слабкі сторони:** доступно тільки англійською мовою, працює тільки з одним видом журналу Windows
Можливості: зростання популярності пошуку аномалій у роботі операційної системи у маленьких компаніях та у індивідуальних користувачів	**Загрози**: конкуренція, структури журналу, поява несумісної версії операційної системи

На основі SWOT-аналізу розроблено альтернативи ринкової поведінки (перелік заходів) для виведення стартап-проекту на ринок та орієнтовний оптимальний час їх ринкової реалізації з огляду на потенційні проекти конкурентів, що можуть бути виведені на ринок (див. табл.4.10, аналіз потенційних конкурентів). Визначені альтернативи проаналізовано з точки зору строків та ймовірності отримання ресурсів (табл.4.14).

Таблиця 4.14 - Альтернативи ринкового впровадження стартап-проекту

№	**Альтернатива (орієнтовний комплекс заходів) ринкової поведінки**	**Ймовірність отримання ресурсів**	**Строки реалізації**
1	Створення застосунку для виявлення аномалій операційної системи Windows та створення попередньо локалізованого інтерфейсу користувача	70%	8 місяці
2	Створення застосунку для виявлення аномалій операційної системи Windows без створення попередньо локалізованого інтерфейсу користувача	30%	5 місяці

Обираємо альтернативу 1.

З означених альтернатив обирається та, для якої: а) отримання ресурсів є більш простим та ймовірним; б) строки реалізації – не набагато більші. Враховуючи, що наявність локалізації збільшить ймовірність отримання ресурсів, то обираємо перший варіант.

4.4 Розроблення ринкової стратегії проекту

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів (табл. 4.15).

Таблиця 4.15 – Вибір цільових груп потенційних споживачів

№ п/п	**Опис профілю цільової групи потенційних клієнтів**	**Готовність споживачів сприйняти продукт**	**Орієнтовний попит в межах цільової групи (сегменту)**	**Інтенсивність конкуренції в сегменті**	**Простота входу у сегмент**
1	Великі продуктові компанії.	Середня: велика конкуренція і можливість власних ІТ-відділів.	Високий.	Велика.	Легко.

Продовженя таблиці 4.15

2	Великі аутсорсові компанії	Середня.	Високий.	Велика.	Середня
3	Маленькі продуктові компанії.	Середня.	Середній.	Середня.	Середня.
4	Маленькі аутсорсові компанії	Низька. Приватні особи воліють продукт за найменшу ціну і не обов’язково якісний.	Низький.	Середня.	Важко.

Як цільові групи обрано усі чотири варіанти.

За результатами аналізу потенційних груп споживачів (сегментів) автори ідеї обирають цільові групи, для яких вони пропонуватимуть свій товар, та визначають стратегію охоплення ринку:

- якщо компанія зосереджується на одному сегменті – вона обирає стратегію концентрованого маркетингу;
- якщо працює із кількома сегментами, розробляючи для них окремо програми ринкового впливу – вона використовує стратегію диференційованого маркетингу;
- якщо компанія працює з усім ринком, пропонуючи стандартизовану програму (включно із характеристиками товару/послуги) – вона використовує масовий маркетинг. Для роботи в обраних сегментах ринку сформовано базову стратегію розвитку (табл. 4.16)

Таблиця 4.16 - Визначення базової стратегії розвитку

№	**Обрана альтернатива розвитку проекту**	**Стратегія охоплення ринку**	**Ключові конкурентоспроможні позиції відповідно до обраної альтернативи**	**Базова стратегія розвитку**
1	Створення застосунку для виявлення аномалій операційної системи попередньо локалізованого інтерфейсу користувача	Ринкове позиціювання	Простота інтерфейсу, відритий доступ до іноземних ринків	Диференціації

Наступним кроком є вибір стратегії конкурентної поведінки (табл. 4.17).

Таблиця 4.17 - Визначення базової стратегії конкурентної поведінки

№	**Чи є проект «першопрохідцем» на ринку?**	**Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?**	**Чи буде компанія копіювати основні характеристики товару конкурента, і які?**	**Стратегія конкурентної поведінки**
1	Ні	Так	Так: базові функції керування ризиками	Зайняття конкурентної ніші

На основі вимог споживачів з обраних сегментів до постачальника (стартап-компанії) та до продукту (див. Табл. 4.6), а також в залежності від обраної базової стратегії розвитку (табл. 4.16) та стратегії конкурентної поведінки (табл. 4.17) розробляється стратегія позиціонування (табл. 4.18). що полягає у формуванні ринкової позиції (комплексу асоціацій), за яким споживачі мають ідентифікувати торгівельну марку/проект.

Таблиця 4.18 - Визначення стратегії позиціонування

№	**Вимоги до товару цільової аудиторії**	**Базова стратегія розвитку**	**Ключові конкурентоспроможні позиції власного стартап-проекту**	**Вибір асоціацій, які мають сформувати комплексну позицію**

				власного проекту (три ключових)
1	Швидкість і зручність роботи, відповідність результатів	Диференціації	локалізованість, висока точність результатів	Швидкість легкість, точність, великі дані, аналітика, операційна система

Результатом виконання підрозділу стала узгоджена система рішень щодо ринкової поведінки стартап-компанії, яка визначає напрями роботи стартап-компанії на ринку.

4.5 Розроблення маркетингової програми стартап-проекту

Першим кроком є формування маркетингової концепції товару, який отримує споживач. Для цього у табл. 4.19 підсумовано результати попереднього аналізу конкурентоспроможності товару. Концепція товару - письмовий опис фізичних та інших характеристик товару, які сприймаються споживачем, і набору вигод, які він обіцяє певній групі споживачів.

Таблиця 4.19 - Визначення ключових переваг концепції потенційного товару

№	**Потреба**	**Вигода, яку пропонує**	**Ключові переваги**

		товар	**перед конкурентами (існуючі або такі, що потрібно створити**
1	Витрачати менше часу виявлення аномалій у роботі операційної системи	Автоматична ідентифікація аномалій у роботі операційної системи	Економія часу та зусиль
2	Можливість слідкувати за активністю користувача за робочим комп'ютером	Можливість виявлення несанкціонованого або незвичного логіну у систему	Контроль роботи користувачів

Розроблена трирівнева маркетингова модель товару: уточнюється ідея продукту та/або послуги, його фізичні складові, особливості процесу його надання (табл. 4.20).

Таблиця 4.20 - Опис трьох рівнів моделі товару

Рівні товару	**Сутність та складові**		
I. Товар за задумом	Товар допомагає користувачам автоматично створювати тривимірні моделі будь-яких обєктів з фотозображень		
II. Товар у реальному виконанні	Хар-ки	М/Нм	Вр/Тх /Тл/Е/Ор
	1) Модуль тривимірної реконструкції моделей 2) Простота у використанні; 3) Можливість розширення	-	-
	Якість: згідно до стандарту ISO 4444 буде проведено тестування		
	Маркування відсутнє.		

III. Товар із підкріпленням	Безкоштовна версія з урізаним функціоналом
	Постійна підтримка для користувачів

1-й рівень - При формуванні задуму товару вирішується питання щодо того, засобом вирішення якої потреби і / або проблеми буде даний товар, яка його основна вигода. Дане питання безпосередньо пов'язаний з формуванням технічного завдання в процесі розробки конструкторської документації на виріб.

2-й рівень - Цей рівень являє рішення того, як буде реалізований товар в реальному/ включає в себе якість, властивості, дизайн, упаковку, ціну.

3-й рівень - Товар з підкріпленням (супроводом) - додаткові послуги та переваги для споживача, що створюються на основі товару за задумом і товару в реальному виконанні (гарантії якості , доставка, умови оплати та ін)

За рахунок чого потенційний товар буде захищено від копіювання: ноу-хау.

Після формування маркетингової моделі товару слід особливо відмітити – чим саме проект буде захищено від копіювання. Захист може бути організовано за рахунок захисту ідеї товару (захист інтелектуальної власності), або ноу-хау, чи комплексне поєднання властивостей і характеристик, закладене на другому та третьому рівнях товару. Наступним кроком визначено цінові межі, якими необхідно керуватись при встановленні ціни на потенційний товар, яке передбачає аналіз ціни на товари-аналоги або товари субститути, а також аналіз рівня доходів цільової групи споживачів (табл. 4.21). Аналіз проводився експертним методом.

Таблиця 4.21 - Визначення меж встановлення ціни

№	**Рівень цін на товари-замінники**	**Рівень цін на товари-аналоги**	**Рівень доходів цільової групи споживачів**	**Верхня та нижня межі встановлення ціни на товар/послугу**

1	1000	1500	200000	500

Наступним кроком є визначення оптимальної системи збуту, в межах якого приймається рішення (табл. 4.22):

- проводити збут власними силами або залучати сторонніх посередників (власна або залучена система збуту);
- вибір та обґрунтування оптимальної глибини каналу збуту;
- вибір та обґрунтування виду посередників.

Таблиця 4.22 - Формування системи збуту

№	**Специфіка закупівельної поведінки цільових клієнтів**	**Функції збуту, які має виконувати постачальник товару**	**Глибина каналу збуту**	**Оптимальна система збуту**
1	Купують ПЗ та роблять щорічні внески для подовження ліцензії	Продаж	1(через посередника)	Власна та через посередників

Останньою складової маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів (табл. 4.23).

Таблиця 4.23 - Концепція маркетингових комунікацій

№	**Специфіка поведінки цільових клієнтів**	**Канали комунікацій, якими користуються цільові клієнти**	**Ключові позиції, обрані для позиціонування**	**Завдання рекламного повідомлення**	**Концепція рекламного звернення**

1	Купівля ліцензій на використання через інтернет повної версії	Інтернет	автоматична ідентифікація ризиків, реєстр ризиків	Показати переваги ПЗ, у тому числі і перед конкурентами	Демо-ролик із використанням

Результатом пункту 5 є ринкова (маркетингова) програма, що включає в себе концепції товару, збуту, просування та попередній аналіз можливостей ціноутворення, спирається на цінності та потреби потенційних клієнтів, конкурентні переваги ідеї, стан та динаміку ринкового середовища, в межах якого впроваджено проект, та відповідну обрану альтернативу ринкової поведінки

Висновки до розділу

Згідно до проведених досліджень:

- існує можливість ринкової комерціалізації проекту;
- існують перспективи впровадження з огляду на потенційні групи клієнтів, бар’єри входження високі, але проект має одну значну перевагу перед конкурентами;
- необхідно реалізувати застосунок для виявлення аномалій роботи операційної системи Windows за технологією C#;
- подальша імплементація є доцільною.

ВИСНОВКИ

У процесі виконання роботи були виконані усі поставлені задачі:

- проведено дослідження сучасних методів та алгоритів застосування згорткових нейронних мереж; проаналізовані алгоритми для ідентифікаії та зіставлення особливих точок (це основа для побудови цифрової тривимірної моделі складної будівлі);
- розроблено архітектуру згорткової нейронної мережі для ідентифікації особливих точок;
- розроблено алгоритм реконструкції тривимірної моделі об’єкту з послідовності зображень (фотографій);
- створено програмний модуль на базі розробленого алгоритмую.

В даному проекті була вирішена проблема відновлення 3D моделей будівель реального світу з невпорядкованих множин зображень. Використання фотограмметрії в поєднанні з реконструкцією за допомогою користувача дає високу якість об'єкта. Основною метою розробленого модуля є демонстрація здійсненності запропонованого методу реконструкції.

Планується подальше вдосконалення, яке полягає у наступному:

- використання різних нейронних мереж, наприклад, конвуляційної НМ, обмеженої машини Больцмана та наївного Баєсевого класифікатора для поліпшеня методу ідентифікації особливих точок;
- модифікація основного алгоритму тривимірної рекнструкції. Наприклад: додати етап кластеризації особливих точок, щоб зникла необхідність допомоги користувача;
- використання нейронних мереж з глибоким навчанням.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Компьютерное зрение – URL: https://ru.wikipedia.org/wiki/ компьютерное_зрение (дата звернення 14.02.2018).
2. Abhishak Yadav, Poonam Yadav. Digital Image Processin*g*. Laxmi Publications, 2009. P. 236
3. Accurate Junction Detection and Characterization in Natural. – URL: https://hal.archives-ouvertes.fr/hal-00631609/document (дата звернення 14.02.2018).
4. V. Rodehorst, A. Koschan. Comparison and evaluation of feature point detectors. *П'ятий міжнародний сімпозіум „Turkish-German Joint Geodetic Days"*. Берлін, 2006.
5. Color histogram – URL: http://en.wikipedia.org/wiki/Color_histogram (дата звернення 14.02.2018).
6. Н. Васильева. Построение и комбинирование признаков в задаче поиска изображений по содержанию. Санкт-Петербург, 2010. С 243.
7. L. David, Object recognition from local scale-invariant features. *Computer Vision and image Understanding*, London. 2008
8. H. Bay, T. Tuytelaars, L. V. Gool, SURF: Speeded Up Robust Features. *Computer Vision and image Understanding*, London. 2008.
9. Y. Ke, R. Sukthankar, PCA-SIFT: A More Distinctive Representation for Local Image Descriptors. URL - https://www.cs.cmu.edu/~rahuls/pub /cvpr2004-keypoint-rahuls.pdf (дата звернення 15.02.2018).
10. Ethan Rublee, Vincent Rabaud, Kurt Konolige, Gary Bradski: "ORB: an efficient alternative to SIFT or SURF", Computer Vision (ICCV), IEEE International Conference on. IEEE, 2011. Pp. 2564 – 2571.
11. X. Yang, K. T. Cheng. LDB: An ultra-fast feature for scalable augmented reality. *IEEE and ACM Intl. Sym. on Mixed and Augmented Reality*, 2012. Pp. 49 – 57.

12. Michael Calonder, Vincent Lepetit, Christoph Strecha, Pascal Fua. Binary Robust Independent Elementary Features. *Твропейская конференция компьютерного(ECCV),* 2010. Pp. 778 – 792.
13. Harris, C., Stephens, M. A Combined Corner and Edge Detector. *Proceedings of the 4th Alvey Vision Conference*, 1988. Pp. 147 – 151.
14. Ethan Rublee, Vincent Rabaud, Kurt Konolige, Gary Bradski. ORB: an efficient alternative to SIFT or SURF. *Computer Vision (ICCV), IEEE International Conference on. IEEE*, 2011. Pp. 2564 – 2571.
15. Stefan Leutenegger, Margarita Chli, Roland Siegwart. BRISK: Binary Robust Invariant Scalable Keypoints. *Computer Vision (ICCV)*, 2011. Pp. 2548 – 2555.
16. S. Grewenig, J. Weickert, C. Schroers, A. Bruhn. Cyclic Schemes for PDEBased Image Analysis. *International Journal of Computer Vision*, 2013
17. J. Weickert, H. Scharr. A scheme for coherence-enhancing diffusion filtering with optimized rotation invariance, *Journal of Visual Communication and Image Representation*, 2002. Pp. 103–118.
18. Herbert Bay, Tinne Tuytelaars, Luc Van Gool. SURF: Speeded Up Robust Features. *Proceedings of the ninth European Conference on Computer Vision*, 2006. Pp. 404 – 417.
19. Академия Intel: Введение в разработку мультимедийных приложений с использованием библиотек OpenCV и IPP. Лекция 3: Детекторы и дескрипторы ключевых точек. Алгоритмы классификации изображений. Задача детектирования объектов на изображениях и методы её решения. – URL: http://www.intuit.ru/studies/courses/10621/1105/lecture/17983?page=1 (дата звернення 14.02.2018)
20. Академия Intel: Введение в разработку мультимедийных приложений с использованием библиотек OpenCV и IPP. Лекция 3: Детекторы и дескрипторы ключевых точек. Алгоритмы классификации изображений. Задача детектирования объектов на изображениях и

методы её решения. – URL: http://www.intuit.ru/studies/courses/10621/1105/lecture/17983?page=1 (дата звернення 14.02.2018)

21. Integral image – URL: http://en.wikipedia.org/wiki/Integral_image.
22. K. G. Derpanis Integral image-based representations. *IEEE Computer Vision and Pattern Recognition*, 2007. Pp. 137 – 141.
23. M. Irani, E. Shechtman. Matchilg Local Self-Similarities across Images and Videos. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007. Pp. 2463 – 2468.
24. Карти Google– URL: https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D1%80%D1%82%D1%8B_Google (дата звернення 14.02.2018)
25. Перегляд вулиць Google Maps – https://www.google.com/streetview/understand (дата звернення 14.02.2018)
26. Application of Feature Point Detection and Matching in 3D Objects Reconstructio– URL: https://www.thinkmind.org/download.php?articleid=patterns_2011_1_40_70070 (дата звернення 14.02.2018)
27. Strecha, C., Hansen, W., Van Gool, L., Fua, P., Thoennessen, U.: On Benchmarking Camera Calibration and Multi-View Stereo for High Resolution Imagery. *CVPR*, 2008. Pp. 423 – 424.
28. Aanaes, H., Dahl, A.L., Pedersen, K.S.: Interesting Interest Points. *IJCV 97*, 2012. Pp. 18–35.
29. Verdie, Y., Yi, K.M., Fua, P., Lepetit, V.: TILDE: A Temporally Invariant Learned Detector. *CVPR*, 2015.

ДОДАТОК А

```
# -*- coding: utf-8 -*-

import math
import numpy as np
import os
import random
import sys
import glob
import cv2
import tensorflow as tf

from spatial_transformer import
inplane_inverse_warp

def inverse_warp_view_2_to_1(heatmaps2,
depths2, depths1, c2Tc1s, K1, K2,
inv_thetas1, thetas2, depth_thresh=0.5,
get_warped_depth=False):
    # compute warping xy coordinate
from view1 to view2
    # Args
    #   depths1: [B,H,W,1] tf.float32
    #   c2Tc1s: [B,4,4] tf.float32
    #   K1,K2: [B,3,3] tf.float32
    #   inv_thetas1: [B,3,3] tf.float32
    #   thetas2: [B,3,3] tf.float32
    # Return
    #   heatmaps1w : [B,H,W,1]
tf.float32 warped heatmaps from camera2
to camera1
    #   visible_masks1 : [B,H,W,1]
tf.float32 visible masks on camera1
    #   xy_u2 : [B,H,W,2] tf.float32
xy-coords-maps from camera1 to camera2

    def norm_meshgrid(batch, height,
width, is_homogeneous=True):
        """Construct a 2D meshgrid.

        Args:
            batch: batch size
            height: height of the grid
            width: width of the grid
            is_homogeneous: whether to
return in homogeneous coordinates
        Returns:
            x,y grid coordinates
[batch, 2 (3 if homogeneous), height,
width]
        """
        x_t =
tf.matmul(tf.ones(shape=tf.stack([heigh
t, 1])),

tf.transpose(tf.expand_dims(
                            tf.linspace(-
1.0, 1.0, width), 1), [1, 0]))
        y_t =
tf.matmul(tf.expand_dims(tf.linspace(-
1.0, 1.0, height), 1),

tf.ones(shape=tf.stack([1, width])))
        if is_homogeneous:
            ones = tf.ones_like(x_t)
            coords = tf.stack([x_t,
y_t, ones], axis=0)
        else:
            coords = tf.stack([x_t,
y_t], axis=0)
        coords =
tf.tile(tf.expand_dims(coords, 0),
[batch, 1, 1, 1])
        return coords

    def norm_xy_coords(xyz, height,
width):
        # xyz: [B,>=3,N], tf.float32
xyz[:,0] = x, xyz[:,1]=y
        # suppose 0<=x<width,
0<=y<height
        # outputs range will be [-1,1]
        x_t = tf.slice(xyz, [0,0,0], [-
1,1,-1])
        y_t = tf.slice(xyz, [0,1,0], [-
1,1,-1])
        z_t = tf.slice(xyz, [0,2,0], [-
1,-1,-1])
        x_t = 2 * (x_t / tf.cast(width-
1,tf.float32)) - 1.0
        y_t = 2 * (y_t /
tf.cast(height-1, tf.float32)) - 1.0
        n_xyz = tf.concat([x_t, y_t,
z_t], axis=1)
        return n_xyz

    def unnorm_xy_coords(xyz, height,
width):
        # xyz: [B,>=3,N], tf.float32
xyz[:,0] = x, xyz[:,1]=y
        # suppose -1<=x<=1, -1<=y<=1
        # outputs range will be
[0,width) or [0,height)

        x_t = tf.slice(xyz, [0,0,0], [-
1,1,-1])
        y_t = tf.slice(xyz, [0,1,0], [-
1,1,-1])
        z_t = tf.slice(xyz, [0,2,0], [-
1,-1,-1])

        x_t = (x_t+1.0) * 0.5 *
tf.cast(width-1, tf.float32)
        y_t = (y_t+1.0) * 0.5 *
tf.cast(height-1, tf.float32)

        u_xyz = tf.concat([x_t, y_t,
z_t], axis=1)
        return u_xyz

    with
tf.name_scope('WarpCoordinates'):
        if K2 is None:
            K2 = K1 # use same
intrinsic matrix

        eps = 1e-6
        batch_size =
tf.shape(depths1)[0]
        height1 = tf.shape(depths1)[1]
        width1 = tf.shape(depths1)[2]

        inv_K1 = tf.matrix_inverse(K1)
```

```
        right_col =
tf.zeros([batch_size,3,1],
dtype=tf.float32)
        bottom_row =
tf.tile(tf.constant([0,0,0,1],
dtype=tf.float32, shape=[1,1,4]),
[batch_size,1,1])
        K2_4x4 = tf.concat([K2,
right_col], axis=2)
        K2_4x4 = tf.concat([K2_4x4,
bottom_row], axis=1)

        xy_n1 =
norm_meshgrid(batch_size, height1,
width1) # [B,3,H,W]
        xy_n1 = tf.reshape(xy_n1,
[batch_size, 3, -1]) # [B,3,N], N=H*W

        # Inverse inplane
transformation on camera1
        if inv_thetas1 is not None:
            xy_n1 =
tf.matmul(inv_thetas1, xy_n1)
            z_n1 = tf.slice(xy_n1,
[0,2,0],[-1,1,-1])
            xy_n1 = xy_n1 / (z_n1+eps)

        # SE(3) transformation : pixel1
to camera1
        Z1 = tf.reshape(depths1,
[batch_size, 1, -1])
        xy_u1 = unnorm_xy_coords(xy_n1,
height=height1, width=width1)
        XYZ1 = tf.matmul(inv_K1, xy_u1)
* Z1
        ones = tf.ones([batch_size, 1,
height1*width1])
        XYZ1 = tf.concat([XYZ1, ones],
axis=1)

        # SE(3) transformation :
camera1 to camera2 to pixel2
        proj_T = tf.matmul(K2_4x4,
c2Tc1s)
        xyz2 = tf.matmul(proj_T, XYZ1)
        z2 = tf.slice(xyz2, [0,2,0], [-
1,1,-1])
        reproj_depths = tf.reshape(z2,
[batch_size, 1, height1, width1])
        reproj_depths =
tf.transpose(reproj_depths,
perm=[0,2,3,1]) # [B,H,W,1]

        xy_u2 = tf.slice(xyz2,
[0,0,0],[-1,3,-1]) / (z2+eps)

        # Inplane transformation on
camera2
        if thetas2 is not None:
            xy_n2 =
norm_xy_coords(xy_u2, height1, width1)
            xy_n2 = tf.matmul(thetas2,
xy_n2)
            z_n2 =
tf.slice(xy_n2,[0,2,0], [-1,1,-1])
            xy_n2 = xy_n2 / (z_n2+eps)
            xy_u2 =
unnorm_xy_coords(xy_n2, height1,
width1)
        xy_u2 = tf.slice(xy_u2,
[0,0,0],[-1,2,-1]) # discard third dim
        xy_u2 = tf.reshape(xy_u2,
[batch_size, 2, height1, width1])
        xy_u2 = tf.transpose(xy_u2,
perm=[0, 2, 3, 1]) # [B,H,W,2]

        heatmaps1w, depths1w =
bilinear_sampling(heatmaps2, xy_u2,
depths2) # it is not correct way to
check depth consistency but it works
well practically
        visible_masks =
get_visibility_mask(xy_u2)
        camfront_masks =
tf.cast(tf.greater(reproj_depths,
tf.zeros((), reproj_depths.dtype)),
tf.float32)
        nonocc_masks =
tf.cast(tf.less(tf.squared_difference(d
epths1w, depths1), depth_thresh**2),
tf.float32)
        visible_masks = visible_masks *
camfront_masks * nonocc_masks # take
logical_and
        heatmaps1w = heatmaps1w *
visible_masks

        if get_warped_depth:
            return heatmaps1w,
visible_masks, xy_u2, depths1w
        else:
            return heatmaps1w,
visible_masks, xy_u2

def get_angle_colorbar():
    hue = np.arange(360)[:,None,
None].astype(np.float32)
    ones = np.ones_like(hue)
    hsv = np.concatenate([hue, ones,
ones], axis=-1)
    colorbar = cv2.cvtColor(hsv,
cv2.COLOR_HSV2RGB)
    colorbar = np.squeeze(colorbar)
    return colorbar

def get_degree_maps(ori_maps):
    # ori_maps : [B,H,W,2], consist of
cos, sin response
    cos_maps = tf.slice(ori_maps,
[0,0,0,0], [-1,-1,-1,1])
    sin_maps = tf.slice(ori_maps,
[0,0,0,1], [-1,-1,-1,1])
    atan_maps = tf.atan2(sin_maps,
cos_maps)
    angle2rgb =
tf.constant(get_angle_colorbar())
    degree_maps =
tf.cast(tf.clip_by_value(atan_maps*180/
np.pi+180, 0, 360), tf.int32)
    degree_maps = tf.gather(angle2rgb,
degree_maps[...,0])
    return degree_maps, atan_maps

# def inverse_warp_view_1_to_2(photos1,
depths1, depths2, c1Tc2s,
intrinsics_3x3, thetas1=None,
thetas2=None, depth_thresh=0.5):
#     if thetas1 is not None:
```

```
#         # inverse in-plane
transformation
#         photo_depths1 =
tf.concat([photos1, depths1], axis=-1)
#         inwarp_photo_depths1, _ =
inplane_inverse_warp(photo_depths1,
thetas1)
#         photos1 =
tf.slice(inwarp_photo_depths1,
[0,0,0,0],[-1,-1,-1,1])
#         depths1 =
tf.slice(inwarp_photo_depths1,
[0,0,0,1],[-1,-1,-1,1])
#     # projective inverse
transformation
#     photos2w, visible_masks2 =
projective_inverse_warp(photos1,
depths2, c1Tc2s,
#
intrinsics_3x3, depths1, depth_thresh)
#     projective_visible_masks2 =
tf.identity(visible_masks2)
#     if thetas2 is not None:
#         # inverse in-plane
transformation
#         photo_masks2 =
tf.concat([photos2w, visible_masks2],
axis=-1)
#         inwarp_photo_masks2, _ =
inplane_inverse_warp(photo_masks2,
thetas2)
#         photos2w =
tf.slice(inwarp_photo_masks2,
[0,0,0,0], [-1,-1,-1,1])
#         visible_masks2 =
tf.slice(inwarp_photo_masks2,
[0,0,0,1], [-1,-1,-1,1])
#     visible_masks2 =
tf.cast(tf.greater(visible_masks2,
0.5), tf.float32)
#     return photos2w, visible_masks2,
projective_visible_masks2

def end_of_frame_masks(height, width,
radius, dtype=tf.float32):
    eof_masks =
tf.ones(tf.stack([1,height-
2*radius,width-2*radius,1]),
dtype=dtype)
    eof_masks = tf.pad(eof_masks,
[[0,0],[radius,radius],[radius,radius],
[0,0]])
    return eof_masks

def morphology_closing(inputs,
dilate_ksize=3, erode_ksize=5):
    curr_in = inputs
    if dilate_ksize > 1:
        curr_in =
tf.nn.max_pool(curr_in,
[1,dilate_ksize,dilate_ksize,1],

strides=[1,1,1,1], padding='SAME')
    if erode_ksize > 1:
        curr_in = -tf.nn.max_pool(-
curr_in, [1,erode_ksize,erode_ksize,1],

strides=[1,1,1,1], padding='SAME')
    return curr_in

def batch_gather_keypoints(inputs,
batch_inds, kpts, xy_order=True):
    # kpts: [N,2] x,y or y,x
    # batch_inds: [N]
    # outputs = inputs[b,y,x]
    if xy_order:
        kp_x, kp_y = tf.split(kpts, 2,
axis=1)
    else:
        kp_y, kp_x = tf.split(kpts, 2,
axis=1)
    if
len(batch_inds.get_shape().as_list())
== 1:
        batch_inds = batch_inds[:,None]
    byx = tf.concat([batch_inds, kp_y,
kp_x], axis=1)
    outputs = tf.gather_nd(inputs, byx)
    return outputs

# def coordinate_se3_warp(kpts1,
batch_inds, intrinsics_3x3, c2Tc1s,
depths1, visible_masks2):
#     # kpts1: [N,2] int32 (x,y)
#     # batch_inds: [N,] int32
[0,batch_size)
#     # intrinsics_3x3: [3,3] float32
#     # c2Tc1s: [B,4,4] float32
#     # depths1: [B,H,W,1] float32
#     with
tf.name_scope('XY_SE3_WARP'):
#         N = tf.shape(kpts1)[0]
#         # gather Z
#         kp_x, kp_y = tf.split(kpts1,
2, axis=1)
#         byx =
tf.concat([batch_inds[:,None], kp_y,
kp_x], axis=1) # [N,3]
#         kp_z = tf.gather_nd(depths1,
byx) # [N,1]

#         # pix2cam
#         pix_kpts =
tf.transpose(tf.cast(kpts1,
tf.float32)) # [2,N]
#         ones = tf.ones([1,N],
dtype=tf.float32)
#         hpix_kpts =
tf.concat([pix_kpts, ones], axis=0) #
[3,N]
#         inv_intrinsics =
tf.matrix_inverse(intrinsics_3x3) #
[3,3]
#         cam_kpts =
tf.matmul(inv_intrinsics, hpix_kpts) *
tf.transpose(kp_z) # [3,3]*[3,N]*[1,N]
= [3,N]
#         hcam_kpts =
tf.concat([cam_kpts, ones], axis=0) #
[4,N]

#         # projection matrix
#         gathered_c2Tc1s =
tf.gather(c2Tc1s, batch_inds) #
[B,4,4],[N] --> [N,4,4]
#         intrinsics_4x4 =
tf.concat([intrinsics_3x3,
tf.zeros([3,1])], axis=1)
#         intrinsics_4x4 =
tf.concat([intrinsics_4x4,
```

```
tf.constant([0.,0.,0.,1.],
shape=[1,4])], axis=0)
#         intrinsics_4x4 =
tf.tile(intrinsics_4x4[None], [N,1,1])
# [N,4,4]
#         projT =
tf.matmul(intrinsics_4x4,
gathered_c2Tc1s) # [N,4,4]

#         # cam2pix
#         hcam_kpts =
tf.transpose(hcam_kpts)[...,None] #
[4,N]->[N,4,1]
#         hcam_kpts_w =
tf.matmul(projT, hcam_kpts)[:,:,0] #
[N,4,4]*[N,4,1]-->[N,4]

#         x_u = tf.slice(hcam_kpts_w,
[0,0],[-1,1])
#         y_u = tf.slice(hcam_kpts_w,
[0,1],[-1,1])
#         z_u = tf.slice(hcam_kpts_w,
[0,2],[-1,1])

#         kp_xw = x_u / (z_u+1e-6)
#         kp_yw = y_u / (z_u+1e-6)
#         # kpts_w = tf.concat([kp_xw,
kp_yw], axis=1) # [N,2]

#         # check visibility
#         x0 = tf.cast(tf.floor(kp_xw),
tf.int32)
#         x1 = x0 + 1
#         y0 = tf.cast(tf.floor(kp_yw),
tf.int32)
#         y1 = y0 + 1

#         height =
tf.shape(visible_masks2)[1]
#         width =
tf.shape(visible_masks2)[2]
#         inside_x =
tf.logical_and(tf.greater_equal(x0, 0),
tf.less(x1, width))
#         inside_y =
tf.logical_and(tf.greater_equal(y0, 0),
tf.less(y1, height))
#         visibility =
tf.cast(tf.logical_and(inside_x,
inside_y), tf.float32)

#         kp_xw_safe =
tf.clip_by_value(tf.cast(tf.round(kp_xw
), tf.int32), 0, width-1)
#         kp_yw_safe =
tf.clip_by_value(tf.cast(tf.round(kp_yw
), tf.int32), 0, height-1)
#         kpts_w_safe =
tf.concat([kp_xw_safe, kp_yw_safe],
axis=1) # [N,1] tf.int32

#         byx =
tf.concat([batch_inds[:,None],
kp_yw_safe, kp_xw_safe], axis=1) #
[N,3]
#         gather_visbility =
tf.gather_nd(visible_masks2, byx) #
[N,1]
#         # print('## ',
visible_masks2.shape, byx.shape,
gather_visbility.shape)
#         visibility = visibility *
gather_visbility # [N,1] tf.float32
#         visibility =
tf.squeeze(visibility, 1)  # [N, ]
tf.float32
#         return kpts_w_safe,
visibility

def coordinate_se3_warp(kpts1,
batch_inds, intrinsics_3x3, c2Tc1s,
depths1, visible_masks2):
    # kpts1: [N,2] int32 (x,y)
    # batch_inds: [N,] int32
[0,batch_size)
    # intrinsics_3x3: [B,3,3] float32
    # c2Tc1s: [B,4,4] float32
    # depths1: [B,H,W,1] float32
    with tf.name_scope('XY_SE3_WARP'):
        N = tf.shape(kpts1)[0]
        # gather Z
        kp_x, kp_y = tf.split(kpts1, 2,
axis=1)
        byx =
tf.concat([batch_inds[:,None], kp_y,
kp_x], axis=1) # [N,3]
        kp_z = tf.gather_nd(depths1,
byx) # [N,1]

        # pix2cam
        pix_kpts =
tf.transpose(tf.cast(kpts1,
tf.float32)) # [2,N]
        ones = tf.ones([1,N],
dtype=tf.float32)
        hpix_kpts =
tf.concat([pix_kpts, ones], axis=0) #
[3,N]
        hpix_kpts =
tf.expand_dims(tf.transpose(hpix_kpts),
axis=-1) # [N,3,1]
        inv_intrinsics =
tf.matrix_inverse(intrinsics_3x3) #
[B,3,3]
        gatherd_inv_intrinsics =
tf.gather(inv_intrinsics, batch_inds) #
[B,3,3],[N]-->[N,3,3]
        cam_kpts =
tf.squeeze(tf.matmul(gatherd_inv_intrin
sics, hpix_kpts), axis=-1)  #
[N,3,3]x[N,3,1] --> [N,3,1] --> [N,3]
        cam_kpts =
tf.transpose(cam_kpts*kp_z) # [3,N]
        hcam_kpts =
tf.concat([cam_kpts, ones], axis=0) #
[4,N]

        # projection matrix
        gathered_c2Tc1s =
tf.gather(c2Tc1s, batch_inds) #
[B,4,4],[N] --> [N,4,4]

        batch =
tf.shape(intrinsics_3x3)[0]
        right_col =
tf.zeros([batch,3,1], dtype=tf.float32)
        bottom_row =
tf.tile(tf.constant([0,0,0,1],
```

```
dtype=tf.float32, shape=[1,1,4]),
[batch,1,1])
        intrinsics_4x4 =
tf.concat([intrinsics_3x3, right_col],
axis=2)
        intrinsics_4x4 =
tf.concat([intrinsics_4x4, bottom_row],
axis=1) # [B,4,4]
        gathered_intrinsics_4x4 =
tf.gather(intrinsics_4x4, batch_inds) #
[B,4,4],[N] --> [N,4,4]

        projT =
tf.matmul(gathered_intrinsics_4x4,
gathered_c2Tc1s) # [N,4,4]

        # cam2pix
        hcam_kpts =
tf.transpose(hcam_kpts)[...,None] #
[4,N]->[N,4,1]
        hcam_kpts_w = tf.matmul(projT,
hcam_kpts)[:,:,0] # [N,4,4]*[N,4,1]--
>[N,4]

        x_u = tf.slice(hcam_kpts_w,
[0,0],[-1,1])
        y_u = tf.slice(hcam_kpts_w,
[0,1],[-1,1])
        z_u = tf.slice(hcam_kpts_w,
[0,2],[-1,1])

        kp_xw = x_u / (z_u+1e-6)
        kp_yw = y_u / (z_u+1e-6)
        # kpts_w = tf.concat([kp_xw,
kp_yw], axis=1) # [N,2]

        # check visibility
        x0 = tf.cast(tf.floor(kp_xw),
tf.int32)
        x1 = x0 + 1
        y0 = tf.cast(tf.floor(kp_yw),
tf.int32)
        y1 = y0 + 1

        height =
tf.shape(visible_masks2)[1]
        width =
tf.shape(visible_masks2)[2]
        inside_x =
tf.logical_and(tf.greater_equal(x0, 0),
tf.less(x1, width))
        inside_y =
tf.logical_and(tf.greater_equal(y0, 0),
tf.less(y1, height))
        visibility =
tf.cast(tf.logical_and(inside_x,
inside_y), tf.float32)

        kp_xw_safe =
tf.clip_by_value(tf.cast(tf.round(kp_xw
), tf.int32), 0, width-1)
        kp_yw_safe =
tf.clip_by_value(tf.cast(tf.round(kp_yw
), tf.int32), 0, height-1)
        kpts_w_safe =
tf.concat([kp_xw_safe, kp_yw_safe],
axis=1) # [N,1] tf.int32

        byx =
tf.concat([batch_inds[:,None],
kp_yw_safe, kp_xw_safe], axis=1) #
[N,3]
        gather_visbility =
tf.gather_nd(visible_masks2, byx) #
[N,1]
        # print('## ',
visible_masks2.shape, byx.shape,
gather_visbility.shape)
        visibility = visibility *
gather_visbility # [N,1] tf.float32
        visibility =
tf.squeeze(visibility, 1)  # [N, ]
tf.float32
        return kpts_w_safe, visibility
def
find_hard_negative_from_myself(feats):
    # feats.shape = [B,K,D]
    K = tf.shape(feats)[1]

    feats1_mat = tf.expand_dims(feats,
axis=2) # [B,K,D] --> [B,K,1,D]
    feats2_mat = tf.expand_dims(feats,
axis=1) # [B,L,D] --> [B,1,L,D]
    feats1_mat = tf.tile(feats1_mat,
[1,1,K,1]) # [B,K,L,D]
    feats2_mat = tf.tile(feats2_mat,
[1,K,1,1]) # [B,K,L,D]

    distances =
tf.reduce_sum(tf.squared_difference(fea
ts1_mat, feats2_mat), axis=-1) #
[B,K,K]
    myself = tf.eye(K)[None] * 1e5
    distances = distances + myself
    min_dist = tf.reduce_min(distances,
axis=2) # min_dist1(i) = min_j
|feats1(i)- feats2(j)|^2 [B,K]
    arg_min = tf.argmin(distances,
axis=2, output_type=tf.int32)

    return min_dist, arg_min, distances

# def
batch_nearest_neighbors_less_memory(fea
ts1, batch_inds1, num_kpts1, feats2,
batch_inds2, num_kpts2, batch_size,
num_parallel=1, back_prop=False):
#     # feats1 = [B*K1, D], feats2 =
[B*K2,D]
#     # batch_inds = [B*K,] takes
[0,batch_size)
#     # num_kpts: [B,] tf.int32
#     # outputs = min_dist, arg_min
[B*K]

#     N1 = tf.shape(feats1)[0]
#     batch_offsets1 =
tf.concat([tf.zeros(1, dtype=tf.int32),
tf.cumsum(num_kpts1)], axis=0)
#     ta_dist1 =
tf.TensorArray(dtype=tf.float32,
size=N1)
#     ta_inds1 =
tf.TensorArray(dtype=tf.int32, size=N1)

#     N2 = tf.shape(feats2)[0]
#     batch_offsets2 =
tf.concat([tf.zeros(1, dtype=tf.int32),
tf.cumsum(num_kpts2)], axis=0)
```

```
#     ta_dist2 =
tf.TensorArray(dtype=tf.float32,
size=N1)
#     ta_inds2 =
tf.TensorArray(dtype=tf.int32, size=N1)

#     init_state = (0, ta_dist1,
ta_inds1, ta_dist2, ta_inds2)
#     condition = lambda i, _, _2: i <
batch_size

#     def body(i, ta_dist, ta_inds):
#         pass


def
find_hard_negative_from_myself_less_mem
ory(feats, batch_inds, num_kpts,
batch_size, num_parallel=1,
back_prop=False):
    # feats = [B*K, D]
    # batch_inds = [B*K,] takes
[0,batch_size)
    # num_kpts: [B,] tf.int32
    # outputs = min_dist, arg_min [B*K]
    N = tf.shape(feats)[0]
    batch_offsets =
tf.concat([tf.zeros(1, dtype=tf.int32),
tf.cumsum(num_kpts)], axis=0)
    ta_dist =
tf.TensorArray(dtype=tf.float32,
size=N)
    ta_inds =
tf.TensorArray(dtype=tf.int32, size=N)
    init_state = (0, ta_dist, ta_inds)
    condition = lambda i, _, _2: i <
batch_size

    def body(i, ta_dist, ta_inds):
        curr_inds =
tf.cast(tf.reshape(tf.where(tf.equal(ba
tch_inds, i)), [-1]), tf.int32)
        is_empty =
tf.equal(tf.size(curr_inds), 0)
        curr_inds = tf.cond(is_empty,
lambda: tf.constant([0,],
dtype=tf.int32), lambda: curr_inds) #
if empty use dammy index but all
results are ignored
        curr_feats = tf.gather(feats,
curr_inds) # [K,D]
        K = tf.shape(curr_feats)[0]
        feats1_mat =
tf.expand_dims(curr_feats, axis=1) #
[K,1,D]
        feats2_mat =
tf.expand_dims(curr_feats, axis=0) #
[1,K,D]
        feats1_mat =
tf.tile(feats1_mat, [1,K,1]) # [K,K,D]
        feats2_mat =
tf.tile(feats2_mat, [K,1,1])
        distances =
tf.reduce_sum(tf.squared_difference(fea
ts1_mat, feats2_mat), axis=-1) # [K,K]
        myself = tf.eye(K) * 1e5
        distances = distances + myself
# avoid to pickup myself
        min_dist =
tf.reduce_min(distances, axis=1) #
min_dist1(i) = min_j |feats1(i)-
feats2(j)|^2 [B,K]
        arg_min = tf.argmin(distances,
axis=1, output_type=tf.int32)

        offset = batch_offsets[i]
        arg_min = arg_min + offset

        ta_dist = tf.cond(is_empty,
lambda: ta_dist, lambda:
ta_dist.scatter(curr_inds, min_dist))
        ta_inds = tf.cond(is_empty,
lambda: ta_inds, lambda:
ta_inds.scatter(curr_inds, arg_min))
        return i+1, ta_dist, ta_inds
#         return i+1,
ta_dist.scatter(curr_inds, min_dist),
ta_inds.scatter(curr_inds, arg_min)

    n, ta_dist_final, ta_inds_final =
tf.while_loop(condition, body,
init_state,

parallel_iterations=num_parallel,

back_prop=back_prop)
    min_dist = ta_dist_final.stack()
    min_inds = ta_inds_final.stack()
    return min_dist, min_inds

def
find_random_hard_negative_from_myself_w
ith_geom_constrain_less_memory(num_pick
up, feats, feats_warp, kpts_warp,
batch_inds, num_kpts, batch_size,
geom_sq_thresh, num_parallel=1,
back_prop=False):
    # find nearest ref-feats index
    # feats = [B*K, D] feature on
image1
    # feats_warp = [B*K, D] feature on
image2 (warped feature from image1)
    # kpts = [B*K, 2] keypoints on
image2 (warped coordinates from image1)
    # geom_sqr_thresh = squre threshold
of x,y coordinate distance
    # batch_inds = [B*K,] takes
[0,batch_size)
    # num_kpts: [B,] tf.int32
    # outputs = min_dist, arg_min [B*K]
    N = tf.shape(feats)[0]
    batch_offsets =
tf.concat([tf.zeros(1, dtype=tf.int32),
tf.cumsum(num_kpts)], axis=0)
    ta_inds =
tf.TensorArray(dtype=tf.int32, size=N)
    init_state = (0, ta_inds)
    condition = lambda i, _: i <
batch_size

    def body(i, ta_inds):
        curr_inds =
tf.cast(tf.reshape(tf.where(tf.equal(ba
tch_inds, i)), [-1]), tf.int32)
        is_empty =
tf.equal(tf.size(curr_inds), 0)
        curr_inds = tf.cond(is_empty,
lambda: tf.constant([0,],
dtype=tf.int32), lambda: curr_inds) #
```

```
if empty use dammy index but all
results are ignored
        curr_feats1 = tf.gather(feats,
curr_inds) # [K,D]
        curr_feats2 =
tf.gather(feats_warp, curr_inds) #
[K,D]
        curr_kpts =
tf.gather(kpts_warp, curr_inds) # [K,2]
        K = tf.shape(curr_feats1)[0]
        feats1_mat =
tf.expand_dims(curr_feats1, axis=1) #
[K,1,D]
        feats2_mat =
tf.expand_dims(curr_feats2, axis=0) #
[1,K,D]
        feats1_mat =
tf.tile(feats1_mat, [1,K,1]) # [K,K,D]
        feats2_mat =
tf.tile(feats2_mat, [K,1,1])
        feat_dists =
tf.reduce_sum(tf.squared_difference(fea
ts1_mat, feats2_mat), axis=-1) # [K,K]
        kp1_mat =
tf.expand_dims(curr_kpts, axis=1) #
[K,1,2]
        kp2_mat =
tf.expand_dims(curr_kpts, axis=0) #
[1,K,2]
        kp1_mat = tf.tile(kp1_mat,
[1,K,1]) # [K,K,2]
        kp2_mat = tf.tile(kp2_mat,
[K,1,1])
        geom_dists =
tf.reduce_sum(tf.squared_difference(kp1
_mat, kp2_mat), axis=-1) # [K,K]
        neighbor_penalty =
tf.cast(tf.less_equal(geom_dists,
geom_sq_thresh), tf.float32) * 1e5
        feat_dists = feat_dists +
neighbor_penalty # avoid to pickup from
neighborhood

        # sort top_k and pickup from
them
        topk_dist, topk_inds =
tf.nn.top_k(-feat_dists, k=num_pickup,
sorted=False) # take the smallest value
        # topk_dist = -topk_dist #
convert comment out because dist are
not necessary
        pickup_inds = tf.concat([

tf.range(K,dtype=tf.int32)[:,None],

tf.random_uniform([K,1], minval=0,
maxval=num_pickup, dtype=tf.int32)
                        ], axis=1)
        min_inds =
tf.gather_nd(topk_inds, pickup_inds)

        offset = batch_offsets[i]
        min_inds = min_inds + offset

        ta_inds = tf.cond(is_empty,
lambda: ta_inds, lambda:
ta_inds.scatter(curr_inds, min_inds))
        return i+1, ta_inds
#         return i+1,
ta_dist.scatter(curr_inds, min_dist),
ta_inds.scatter(curr_inds, arg_min)

    n, ta_inds_final =
tf.while_loop(condition, body,
init_state,

parallel_iterations=num_parallel,

back_prop=back_prop)
    min_inds = ta_inds_final.stack()
    return min_inds

def
find_hard_negative_from_myself_with_geo
m_constrain_less_memory(feats,
feats_warp, kpts_warp, batch_inds,
num_kpts, batch_size, geom_sq_thresh,
num_parallel=1, back_prop=False):
    # find nearest ref-feats index
    # feats = [B*K, D] feature on
image1
    # feats_warp = [B*K, D] feature on
image2 (warped feature from image1)
    # kpts = [B*K, 2] keypoints on
image2 (warped coordinates from image1)
    # geom_sqr_thresh = squre threshold
of x,y coordinate distance
    # batch_inds = [B*K,] takes
[0,batch_size)
    # num_kpts: [B,] tf.int32
    # outputs = min_dist, arg_min [B*K]
    N = tf.shape(feats)[0]
    batch_offsets =
tf.concat([tf.zeros(1, dtype=tf.int32),
tf.cumsum(num_kpts)], axis=0)
    ta_dist =
tf.TensorArray(dtype=tf.float32,
size=N)
    ta_inds =
tf.TensorArray(dtype=tf.int32, size=N)
    init_state = (0, ta_dist, ta_inds)
    condition = lambda i, _, _2: i <
batch_size

    def body(i, ta_dist, ta_inds):
        curr_inds =
tf.cast(tf.reshape(tf.where(tf.equal(ba
tch_inds, i)), [-1]), tf.int32)
        is_empty =
tf.equal(tf.size(curr_inds), 0)
        curr_inds = tf.cond(is_empty,
lambda: tf.constant([0,],
dtype=tf.int32), lambda: curr_inds) #
if empty use dammy index but all
results are ignored
        curr_feats1 = tf.gather(feats,
curr_inds) # [K,D]
        curr_feats2 =
tf.gather(feats_warp, curr_inds) #
[K,D]
        curr_kpts =
tf.gather(kpts_warp, curr_inds) # [K,2]
        K = tf.shape(curr_feats1)[0]
        feats1_mat =
tf.expand_dims(curr_feats1, axis=1) #
[K,1,D]
```

```
        feats2_mat =
tf.expand_dims(curr_feats2, axis=0) #
[1,K,D]
        feats1_mat =
tf.tile(feats1_mat, [1,K,1]) # [K,K,D]
        feats2_mat =
tf.tile(feats2_mat, [K,1,1])
        feat_dists =
tf.reduce_sum(tf.squared_difference(fea
ts1_mat, feats2_mat), axis=-1) # [K,K]
        kp1_mat =
tf.expand_dims(curr_kpts, axis=1) #
[K,1,2]
        kp2_mat =
tf.expand_dims(curr_kpts, axis=0) #
[1,K,2]
        kp1_mat = tf.tile(kp1_mat,
[1,K,1]) # [K,K,2]
        kp2_mat = tf.tile(kp2_mat,
[K,1,1])
        geom_dists =
tf.reduce_sum(tf.squared_difference(kp1
_mat, kp2_mat), axis=-1) # [K,K]
        neighbor_penalty =
tf.cast(tf.less_equal(geom_dists,
geom_sq_thresh), tf.float32) * 1e5
        feat_dists = feat_dists +
neighbor_penalty # avoid to pickup from
neighborhood
        min_dist =
tf.reduce_min(feat_dists, axis=1) #
min_dist1(i) = min_j |feats1(i)-
feats2(j)|^2 [B,K]
        arg_min = tf.argmin(feat_dists,
axis=1, output_type=tf.int32) # find
closest warped feats by using
argmin(axis=1)

        offset = batch_offsets[i]
        arg_min = arg_min + offset

        ta_dist = tf.cond(is_empty,
lambda: ta_dist, lambda:
ta_dist.scatter(curr_inds, min_dist))
        ta_inds = tf.cond(is_empty,
lambda: ta_inds, lambda:
ta_inds.scatter(curr_inds, arg_min))
        return i+1, ta_dist, ta_inds
#         return i+1,
ta_dist.scatter(curr_inds, min_dist),
ta_inds.scatter(curr_inds, arg_min)

    n, ta_dist_final, ta_inds_final =
tf.while_loop(condition, body,
init_state,

parallel_iterations=num_parallel,

back_prop=back_prop)
    min_dist = ta_dist_final.stack()
    min_inds = ta_inds_final.stack()
    return min_dist, min_inds

def
imperfect_find_hard_negative_from_mysel
f_with_geom_constrain_less_memory(feats
, kpts, batch_inds, num_kpts,
batch_size, geom_sq_thresh,
num_parallel=1, back_prop=False):
    # feats = [B*K, D]
    # kpts = [B*K, 2]
    # geom_sqr_thresh = squre threshold
of x,y coordinate distance
    # batch_inds = [B*K,] takes
[0,batch_size)
    # num_kpts: [B,] tf.int32
    # outputs = min_dist, arg_min [B*K]
    N = tf.shape(feats)[0]
    batch_offsets =
tf.concat([tf.zeros(1, dtype=tf.int32),
tf.cumsum(num_kpts)], axis=0)
    ta_dist =
tf.TensorArray(dtype=tf.float32,
size=N)
    ta_inds =
tf.TensorArray(dtype=tf.int32, size=N)
    init_state = (0, ta_dist, ta_inds)
    condition = lambda i, _, _2: i <
batch_size

    def body(i, ta_dist, ta_inds):
        curr_inds =
tf.cast(tf.reshape(tf.where(tf.equal(ba
tch_inds, i)), [-1]), tf.int32)
        is_empty =
tf.equal(tf.size(curr_inds), 0)
        curr_inds = tf.cond(is_empty,
lambda: tf.constant([0,],
dtype=tf.int32), lambda: curr_inds) #
if empty use dammy index but all
results are ignored
        curr_feats = tf.gather(feats,
curr_inds) # [K,D]
        curr_kpts = tf.gather(kpts,
curr_inds) # [K,2]
        K = tf.shape(curr_feats)[0]
        feats1_mat =
tf.expand_dims(curr_feats, axis=1) #
[K,1,D]
        feats2_mat =
tf.expand_dims(curr_feats, axis=0) #
[1,K,D]
        feats1_mat =
tf.tile(feats1_mat, [1,K,1]) # [K,K,D]
        feats2_mat =
tf.tile(feats2_mat, [K,1,1])
        feat_dists =
tf.reduce_sum(tf.squared_difference(fea
ts1_mat, feats2_mat), axis=-1) # [K,K]
        kp1_mat =
tf.expand_dims(curr_kpts, axis=1) #
[K,1,2]
        kp2_mat =
tf.expand_dims(curr_kpts, axis=0) #
[1,K,2]
        kp1_mat = tf.tile(kp1_mat,
[1,K,1]) # [K,K,2]
        kp2_mat = tf.tile(kp2_mat,
[K,1,1])
        geom_dists =
tf.reduce_sum(tf.squared_difference(kp1
_mat, kp2_mat), axis=-1) # [K,K]
        neighbor_penalty =
tf.cast(tf.less_equal(geom_dists,
geom_sq_thresh), tf.float32) * 1e5
        feat_dists = feat_dists +
neighbor_penalty # avoid to pickup from
neighborhood
        min_dist =
tf.reduce_min(feat_dists, axis=1) #
```

```
min_dist1(i) = min_j |feats1(i)-
feats2(j)|^2 [B,K]
        arg_min = tf.argmin(feat_dists,
axis=1, output_type=tf.int32)

        offset = batch_offsets[i]
        arg_min = arg_min + offset

        ta_dist = tf.cond(is_empty,
lambda: ta_dist, lambda:
ta_dist.scatter(curr_inds, min_dist))
        ta_inds = tf.cond(is_empty,
lambda: ta_inds, lambda:
ta_inds.scatter(curr_inds, arg_min))
        return i+1, ta_dist, ta_inds
#         return i+1,
ta_dist.scatter(curr_inds, min_dist),
ta_inds.scatter(curr_inds, arg_min)

    n, ta_dist_final, ta_inds_final =
tf.while_loop(condition, body,
init_state,

parallel_iterations=num_parallel,

back_prop=back_prop)
    min_dist = ta_dist_final.stack()
    min_inds = ta_inds_final.stack()
    return min_dist, min_inds

def
find_random_negative_from_myself_less_m
emory(feats, batch_inds, num_kpts,
batch_size, num_parallel=1,
back_prop=False):
    # feats = [B*K, D]
    # batch_inds = [B*K,] takes
[0,batch_size)
    # num_kpts: [B,] tf.int32
    # outputs = min_dist, arg_min [B*K]
    N = tf.shape(feats)[0]
    batch_offsets =
tf.concat([tf.zeros(1, dtype=tf.int32),
tf.cumsum(num_kpts)], axis=0)
    ta_inds =
tf.TensorArray(dtype=tf.int32, size=N)
    init_state = (0, ta_inds)
    condition = lambda i, _,: i <
batch_size

    def body(i, ta_inds):
        curr_inds =
tf.cast(tf.reshape(tf.where(tf.equal(ba
tch_inds, i)), [-1]), tf.int32)
        is_empty =
tf.equal(tf.size(curr_inds), 0)
        curr_inds = tf.cond(is_empty,
lambda: tf.constant([0,],
dtype=tf.int32), lambda: curr_inds) #
if empty use dammy index but all
results are ignored
        curr_feats = tf.gather(feats,
curr_inds) # [K,D]
        K = tf.shape(curr_feats)[0]
        # could be select itself in
case K=1
        rnd_inds =
tf.random_uniform([K], minval=1,
maxval=tf.maximum(K,2), dtype=tf.int32)
# random_uniform doesn't allow the case
minval==maxval
#         rnd_inds = tf.ones([K],
dtype=tf.int32) * 2
        rnd_inds = tf.range(K) +
rnd_inds
        rnd_inds =
tf.where(tf.less(rnd_inds, K),
rnd_inds, rnd_inds-K)
        rnd_inds = rnd_inds +
batch_offsets[i]

        ta_inds = tf.cond(is_empty,
lambda: ta_inds, lambda:
ta_inds.scatter(curr_inds, rnd_inds))
        return i+1, ta_inds

    n, ta_inds_final =
tf.while_loop(condition, body,
init_state,

parallel_iterations=num_parallel,

back_prop=back_prop)
    rnd_inds = ta_inds_final.stack()
    return rnd_inds


# def find_correspondences(points,
indices):
#     # points: tf.float32, [B,K,2]
#     # indices: tf.int32, [B,L]
indices[i,j] = [0,K)
#     B = tf.shape(indices)[0]
#     L = tf.shape(indices)[1]

#     batch_indices =
tf.tile(tf.range(B,
dtype=indices.dtype)[:,None], [1,L])
#     indices = tf.reshape(indices, [-
1,1])
#     batch_indices =
tf.reshape(batch_indices, [-1,1])
#     indices =
tf.concat([batch_indices, indices],
axis=1) # [B*L,2]

#     gathered_points =
tf.gather_nd(points, indices)
#     gathered_points =
tf.reshape(gathered_points, [B,L,-1])
#     return gathered_points

# def batch_nearest_neighbors(feats1,
feats2):
#     # feats1.shape = [B,K,D]
#     # feats2.shape = [B,L,D]
#     K = tf.shape(feats1)[1]
#     L = tf.shape(feats2)[1]

#     feats1_mat =
tf.expand_dims(feats1, axis=2) #
[B,K,D] --> [B,K,1,D]
#     feats2_mat =
tf.expand_dims(feats2, axis=1) #
[B,L,D] --> [B,1,L,D]
#     feats1_mat = tf.tile(feats1_mat,
[1,1,L,1]) # [B,K,L,D]
#     feats2_mat = tf.tile(feats2_mat,
[1,K,1,1]) # [B,K,L,D]
```

```
#     distances =
tf.reduce_sum(tf.squared_difference(fea
ts1_mat, feats2_mat), axis=-1) #
[B,K,L]
#     min_dist1 =
tf.reduce_min(distances, axis=2) #
min_dist1(i) = min_j |feats1(i)-
feats2(j)|^2 [B,K]
#     arg_min1 = tf.argmin(distances,
axis=2, output_type=tf.int32)
#     min_dist2 =
tf.reduce_min(distances, axis=1) #
[B,L]
#     arg_min2 = tf.argmin(distances,
axis=1, output_type=tf.int32)

#     return min_dist1, arg_min1,
min_dist2, arg_min2

def nearest_neighbors(feats1, feats2):
    # feats1.shape = [K,D]
    # feats2.shape = [L,D]
    K = tf.shape(feats1)[0]
    L = tf.shape(feats2)[0]

    feats1_mat = tf.expand_dims(feats1,
axis=1) # [K,D] --> [K,1,D]
    feats2_mat = tf.expand_dims(feats2,
axis=0) # [K,D] --> [1,L,D]
    feats1_mat = tf.tile(feats1_mat,
[1,L,1]) # [K,L,D]
    feats2_mat = tf.tile(feats2_mat,
[K,1,1]) # [K,L,D]

    distances =
tf.reduce_sum(tf.squared_difference(fea
ts1_mat, feats2_mat), axis=-1)
    min_dist1 =
tf.reduce_min(distances, axis=1) #
min_dist1(i) = min_j |feats1(i)-
feats2(j)|^2
    arg_min1 = tf.argmin(distances,
axis=1)
    min_dist2 =
tf.reduce_min(distances, axis=0)
    arg_min2 = tf.argmin(distances,
axis=0)

    return min_dist1, arg_min1,
min_dist2, arg_min2, distances


# def
nearest_neighbors_less_memory(feats1,
feats2, back_prop=False,
num_parallel=10):
#     # min_dist(i) = min_j |feats1(i)-
feats(j|**2
#     # arg_min(i) = arg min_j
|feats1(i)-feats(j|**2
#     # Less memory but too slow on
notebook but why ?
#     N = tf.shape(feats1)[0]
#     ta_dist =
tf.TensorArray(dtype=tf.float32,
size=N)
#     ta_inds =
tf.TensorArray(dtype=tf.int32, size=N)
#     init_state = (0, ta_dist,
ta_inds)
#     condition = lambda i, _, _2: i <
N

#     def body(i, ta_dist, ta_inds):
#         dists = tf.reduce_sum(
tf.squared_difference(feats1[i],
feats2), axis=1)
#         min_dist =
tf.reduce_min(dists)
#         min_inds =
tf.cast(tf.argmin(dists), tf.int32)
#         return i+1, ta_dist.write(i,
min_dist), ta_inds.write(i, min_inds)

#     n, ta_dist_final, ta_inds_final =
tf.while_loop(condition, body,
init_state,
#
parallel_iterations=num_parallel,
back_prop=back_prop)

#     min_dist = ta_dist_final.stack()
#     min_inds = ta_inds_final.stack()

#     return min_dist, min_inds

def extract_keypoints(top_k):
    coords = tf.where(tf.greater(top_k,
0.))
    num_kpts = tf.reduce_sum(top_k,
axis=[1,2,3])
    coords = tf.cast(coords, tf.int32)
    batch_inds, kp_y, kp_x, _ =
tf.split(coords, 4, axis=-1)
    batch_inds = tf.reshape(batch_inds,
[-1])
    kpts = tf.concat([kp_x, kp_y],
axis=1)

    num_kpts = tf.cast(num_kpts,
tf.int32)
    # kpts: [N,2] (N=B*K)
    # batch_inds: N,
    # num_kpts: B
    return kpts, batch_inds, num_kpts

def
extract_patches_from_keypoints(feat_map
s, kpts, batch_inds, crop_radius,
patch_size):
    # feat_maps: [B,H,W,C]
    # kpts: [N,2]
    # batch_inds: [N,]
    # crop_radius, patch_size: scalar

    patch_size = (patch_size,
patch_size)
    with
tf.name_scope('extract_patches_from_key
points'):
        kp_x, kp_y = tf.split(kpts, 2,
axis=-1)

        bboxes =
tf.cast(tf.concat([kp_y-crop_radius,
kp_x-crop_radius, kp_y+crop_radius,
kp_x+crop_radius],
```

```
                                axis=1),
tf.float32)# [num_boxes, 4]
        # normalize bounding boxes
        height =
tf.cast(tf.shape(feat_maps)[1],
tf.float32)
        width =
tf.cast(tf.shape(feat_maps)[2],
tf.float32)
        inv_imsizes =
tf.stack([1./height, 1./width,
1./height, 1./width])[None] # [1,4]
        bboxes = bboxes * inv_imsizes

        crop_patches =
tf.image.crop_and_resize(feat_maps,
bboxes, batch_inds, patch_size)


        return crop_patches

# def
extract_patches_from_keypoints(feat_map
s, top_k, crop_radius, patch_size):
#     patch_size = (patch_size,
patch_size)
#     with
tf.name_scope('extract_patches_from_key
points'):
#         coords =
tf.where(tf.greater(top_k, 0.))
#         coords = tf.cast(coords,
tf.int32)
#         box_ind, kp_y, kp_x, _ =
tf.split(coords, 4, axis=-1)
#         box_ind = tf.reshape(box_ind,
[-1])

#         kpts = tf.concat([kp_x,
kp_y], axis=1)

#         bboxes =
tf.cast(tf.concat([kp_y-crop_radius,
kp_x-crop_radius, kp_y+crop_radius,
kp_x+crop_radius],
#                                axis=1),
tf.float32)# [num_boxes, 4]
#         # normalize bounding boxes
#         height =
tf.cast(tf.shape(feat_maps)[1],
tf.float32)
#         width =
tf.cast(tf.shape(feat_maps)[2],
tf.float32)
#         inv_imsizes =
tf.stack([1./height, 1./width,
1./height, 1./width])[None] # [1,4]
#         bboxes = bboxes * inv_imsizes

#         crop_patches =
tf.image.crop_and_resize(feat_maps,
bboxes, box_ind, patch_size)


#         return crop_patches, kpts,
box_ind

def make_intrinsics_3x3(fx, fy, cx,
cy):
    # non-batch inputs, outputs
    # inputs are scalar, output is 3x3
tensor
    r1 = tf.expand_dims(tf.stack([fx,
0., cx]), axis=0) # [1,3]
    r2 = tf.expand_dims(tf.stack([0.,
fy, cy]), axis=0)
    r3 = tf.constant([0.,0.,1.],
shape=[1, 3])
    intrinsics = tf.concat([r1, r2,
r3], axis=0)
    return intrinsics

def get_gauss_filter_weight(ksize,
sig):
    mu_x = mu_y = ksize//2
    if sig == 0:
        psf = np.zeros((ksize, ksize),
dtype=np.float32)
        psf[mu_y,mu_x] = 1.0
    else:
        xy = np.indices((ksize,ksize))
        x = xy[1,:,:]
        y = xy[0,:,:]
        psf  = np.exp(-((x-
mu_x)**2/(2*sig**2) + (y-
mu_y)**2/(2*sig**2)))
    return psf

def spatial_softmax(logits, ksize,
com_strength=1.0):

    max_logits = tf.nn.max_pool(logits,
[1,ksize,ksize, 1],

strides=[1,1,1,1], padding='SAME')
    sum_filter =
tf.constant(np.ones((ksize,ksize,1,1)),
dtype=tf.float32)
    ex = tf.exp(com_strength * (logits
- max_logits))
    sum_ex = tf.nn.conv2d(ex,
sum_filter, [1,1,1,1], padding='SAME')
    probs = ex / (sum_ex + 1e-6)
    return probs

def soft_nms_3d(scale_logits, ksize,
com_strength=1.0):
    # apply softmax on scalespace
logits
    # scale_logits: [B,H,W,S]
    num_scales =
scale_logits.get_shape().as_list()[-1]

    scale_logits_d =
tf.transpose(scale_logits[...,None],
[0,3,1,2,4]) # [B,S,H,W,1] in order to
apply pool3d
    max_maps =
tf.nn.max_pool3d(scale_logits_d,
[1,num_scales,ksize,ksize,1],
[1,num_scales,1,1,1], padding='SAME')
    max_maps =
tf.transpose(max_maps[...,0],
[0,2,3,1]) # [B,H,W,S]
    exp_maps = tf.exp(com_strength *
(scale_logits-max_maps))
    exp_maps_d =
tf.transpose(exp_maps[...,None],
[0,3,1,2,4]) # [B,S,H,W,1]
```

```
    sum_filter =
tf.constant(np.ones((num_scales, ksize,
ksize, 1, 1)), dtype=tf.float32)
    sum_ex = tf.nn.conv3d(exp_maps_d,
sum_filter, [1,num_scales,1,1,1],
padding='SAME')
    sum_ex =
tf.transpose(sum_ex[...,0], [0,2,3,1])
# [B,H,W,S]
    probs = exp_maps / (sum_ex + 1e-6)

    return probs

def instance_normalization(inputs):
    # normalize 0-means 1-variance in
each sample (not take batch-axis)
    inputs_dim =
inputs.get_shape().ndims
    # Epsilon to be used in the
tf.nn.batch_normalization
    var_eps = 1e-3
    if inputs_dim == 4:
        moments_dims = [1,2] # NHWC
format
    elif inputs_dim == 2:
        moments_dims = [1]
    else:
        raise
ValueError('instance_normalization
suppose input dim is 4:
inputs_dim={}\n'.format(inputs_dim))

    mean, variance =
tf.nn.moments(inputs,
axes=moments_dims, keep_dims=True)
    outputs =
tf.nn.batch_normalization(inputs, mean,
variance, None, None, var_eps) # non-
parametric normalization
    return outputs


def non_max_suppression(inputs,
thresh=0.0, ksize=3, dtype=tf.float32,
name='NMS'):
    with tf.name_scope(name): # add
namespace to keep graph clean
        dtype = inputs.dtype
        batch = tf.shape(inputs)[0]
        height = tf.shape(inputs)[1]
        width = tf.shape(inputs)[2]
        channel = tf.shape(inputs)[3]
        hk = ksize // 2
        zeros = tf.zeros_like(inputs)
        works =
tf.where(tf.less(inputs, thresh),
zeros, inputs)
        works_pad = tf.pad(works,
[[0,0], [2*hk,2*hk], [2*hk,2*hk],
[0,0]], mode='CONSTANT')
        map_augs = []
        for i in range(ksize):
            for j in range(ksize):
                curr_in =
tf.slice(works_pad, [0, i, j, 0], [-1,
height+2*hk, width+2*hk, -1])

map_augs.append(curr_in)
        num_map = len(map_augs) #
ksize*ksize
        center_map =
map_augs[num_map//2]
        peak_mask =
tf.greater(center_map, map_augs[0])
        for n in range(1, num_map):
            if n == num_map // 2:
                continue
            peak_mask =
tf.logical_and(peak_mask,
tf.greater(center_map, map_augs[n]))
        peak_mask = tf.slice(peak_mask,
[0,hk,hk,0],[-1,height,width,-1])
        if dtype != tf.bool:
            peak_mask =
tf.cast(peak_mask, dtype=dtype)

peak_mask.set_shape(inputs.shape) #
keep shape information
        return peak_mask

# def
make_top_k_sparse_tensor(heatmaps,
k=256):
#     batch_size =
tf.shape(heatmaps)[0]
#     _, height, width, channel =
heatmaps.get_shape().as_list()
#     heatmaps_flt =
tf.reshape(heatmaps, [batch_size, -1])
#     values, indices =
tf.nn.top_k(heatmaps_flt, k=k,
sorted=False)
#     boffset =
tf.expand_dims(tf.range(batch_size) *
width * height, axis=1)
#     indices = indices + boffset
#     indices = tf.reshape(indices, [-
1])
#     top_k_maps =
tf.sparse_to_dense(indices,
[batch_size*height*width*channel], 1,
0, validate_indices=False)
#     top_k_maps =
tf.reshape(top_k_maps, [batch_size,
height, width, 1])
#     return tf.cast(top_k_maps,
tf.float32)

def make_top_k_sparse_tensor(heatmaps,
k=256, get_kpts=False):
    batch_size = tf.shape(heatmaps)[0]
    height = tf.shape(heatmaps)[1]
    width = tf.shape(heatmaps)[2]
    heatmaps_flt = tf.reshape(heatmaps,
[batch_size, -1])
    imsize = tf.shape(heatmaps_flt)[1]

    values, xy_indices =
tf.nn.top_k(heatmaps_flt, k=k,
sorted=False)
    boffset =
tf.expand_dims(tf.range(batch_size) *
imsize, axis=1)
    indices = xy_indices + boffset
    indices = tf.reshape(indices, [-1])
    top_k_maps =
tf.sparse_to_dense(indices,
```

```
[batch_size*imsize], 1, 0,
validate_indices=False)
    top_k_maps = tf.reshape(top_k_maps,
[batch_size, height, width, 1])
    top_k_maps = tf.cast(top_k_maps,
tf.float32)
    if get_kpts:
        kpx = tf.mod(xy_indices, width)
        kpy = xy_indices // width
        batch_inds =
tf.tile(tf.range(batch_size,
dtype=tf.int32)[:,None], [1,k])
        kpts =
tf.concat([tf.reshape(kpx, [-1,1]),
tf.reshape(kpy, [-1,1])], axis=1) #
B*K,2
        batch_inds =
tf.reshape(batch_inds, [-1])
        num_kpts =
tf.ones([batch_size], dtype=tf.int32) *
k
        return top_k_maps, kpts,
batch_inds, num_kpts
    else:
        return top_k_maps


def d_softargmax(d_heatmaps,
block_size, com_strength=10):
    # d_heatmaps = [batch, height/N,
width/N, N**2]
    # fgmask = [batch, height/N,
width/N]
    pos_array_x, pos_array_y =
tf.meshgrid(tf.range(block_size),
tf.range(block_size))
    pos_array_x =
tf.cast(tf.reshape(pos_array_x, [-1]),
tf.float32)
    pos_array_y =
tf.cast(tf.reshape(pos_array_y, [-1]),
tf.float32)

    max_out = tf.reduce_max(
        d_heatmaps, axis=-1,
keep_dims=True)
    o = tf.exp(com_strength *
(d_heatmaps - max_out))  # + eps
    sum_o = tf.reduce_sum(
        o, axis=-1, keep_dims=True)
    x = tf.reduce_sum(
        o * tf.reshape(pos_array_x, [1,
1, 1, -1]),
        axis=-1, keep_dims=True
    ) / sum_o
    y = tf.reduce_sum(
        o * tf.reshape(pos_array_y, [1,
1, 1, -1]),
        axis=-1, keep_dims=True
    ) / sum_o

    # x,y shape = [B,H,W,1]
    coords = tf.concat([x, y], axis=-1)
    # x = tf.squeeze(x, axis=-1)
    # y = tf.squeeze(y, axis=-1)

    return coords

def softargmax(score_map,
com_strength=10):
    # out.shape = [batch, height,
width, 1]
    height = tf.shape(score_map)[1]
    width = tf.shape(score_map)[2]
    md = len(score_map.shape)
    # CoM to get the coordinates
    pos_array_x =
tf.cast(tf.range(height),
dtype=tf.float32)
    pos_array_y =
tf.cast(tf.range(height),
dtype=tf.float32)

    max_out = tf.reduce_max(
        score_map, axis=list(range(1,
md)), keep_dims=True)
    o = tf.exp(com_strength *
(score_map - max_out))  # + eps
    sum_o = tf.reduce_sum(
        o, axis=list(range(1, md)),
keep_dims=True)
    x = tf.reduce_sum(
        o * tf.reshape(pos_array_x, [1,
1, -1, 1]),
        axis=list(range(1, md)),
keep_dims=True
    ) / sum_o
    y = tf.reduce_sum(
        o * tf.reshape(pos_array_y, [1,
-1, 1, 1]),
        axis=list(range(1, md)),
keep_dims=True
    ) / sum_o

    # Remove the unecessary dimensions
(i.e. flatten them)
    x = tf.reshape(x, (-1,))
    y = tf.reshape(y, (-1,))

    return x, y

##--------------------------
## Bilinear Inverse Warping
##--------------------------

def meshgrid(batch, height, width,
is_homogeneous=True):
    """Construct a 2D meshgrid.

    Args:
        batch: batch size
        height: height of the grid
        width: width of the grid
        is_homogeneous: whether to
return in homogeneous coordinates
    Returns:
        x,y grid coordinates [batch, 2
(3 if homogeneous), height, width]
    """
    x_t =
tf.matmul(tf.ones(shape=tf.stack([heigh
t, 1])),

tf.transpose(tf.expand_dims(
                        tf.linspace(-1.0,
1.0, width), 1), [1, 0]))
    y_t =
tf.matmul(tf.expand_dims(tf.linspace(-
1.0, 1.0, height), 1),
```

```
tf.ones(shape=tf.stack([1, width])))
    x_t = (x_t + 1.0) * 0.5 *
tf.cast(width - 1, tf.float32)
    y_t = (y_t + 1.0) * 0.5 *
tf.cast(height - 1, tf.float32)
    if is_homogeneous:
        ones = tf.ones_like(x_t)
        coords = tf.stack([x_t, y_t,
ones], axis=0)
    else:
        coords = tf.stack([x_t, y_t],
axis=0)
    coords =
tf.tile(tf.expand_dims(coords, 0),
[batch, 1, 1, 1])
    return coords

# def pixel2cam(depths, pixel_coords,
inv_intrinsics_3x3,
is_homogeneous=True):
#     """Transforms coordinates in the
pixel frame to the camera frame.

#     Args:
#         depths: [batch, height,
width]
#         pixel_coords: homogeneous
pixel coordinates [batch, 3, height,
width]
#         intrinsics: camera intrinsics
[batch, 3, 3]
#         is_homogeneous: return in
homogeneous coordinates
#     Returns:
#         Coords in the camera frame
[batch, 3 (4 if homogeneous), height,
width]
#     """
#     batch = tf.shape(depths)[0]
#     height = tf.shape(depths)[1]
#     width = tf.shape(depths)[2]

#     depths = tf.reshape(depths,
[batch, 1, -1])
#     pixel_coords =
tf.reshape(pixel_coords, [batch, 3, -
1])
#     inv_intrinsics =
tf.tile(tf.expand_dims(inv_intrinsics_3
x3, axis=0), [batch, 1, 1])
#     cam_coords =
tf.matmul(inv_intrinsics, pixel_coords)
* depths
#     if is_homogeneous:
#         ones = tf.ones([batch, 1,
height*width])
#         cam_coords =
tf.concat([cam_coords, ones], axis=1)
#     cam_coords =
tf.reshape(cam_coords, [batch, -1,
height, width])
#     return cam_coords
def pixel2cam(depths, pixel_coords,
inv_intrinsics, is_homogeneous=True):
    """Transforms coordinates in the
pixel frame to the camera frame.

    Args:
        depths: [batch, height, width]
        pixel_coords: homogeneous pixel
coordinates [batch, 3, height, width]
        intrinsics: camera intrinsics
[batch, 3, 3]
        is_homogeneous: return in
homogeneous coordinates
    Returns:
        Coords in the camera frame
[batch, 3 (4 if homogeneous), height,
width]
    """
    batch = tf.shape(depths)[0]
    height = tf.shape(depths)[1]
    width = tf.shape(depths)[2]

    depths = tf.reshape(depths, [batch,
1, -1])
    pixel_coords =
tf.reshape(pixel_coords, [batch, 3, -
1])
    cam_coords =
tf.matmul(inv_intrinsics, pixel_coords)
* depths

    # if thetas is not None:
    #     inv_thetas =
tf.matrix_inverse(thetas)
    #     cam_coords =
tf.matmul(inv_thetas, cam_coords)
    if is_homogeneous:
        ones = tf.ones([batch, 1,
height*width])
        cam_coords =
tf.concat([cam_coords, ones], axis=1)
    cam_coords = tf.reshape(cam_coords,
[batch, -1, height, width])
    return cam_coords

def cam2pixel(cam_coords, proj):
    """Transforms coordinates in a
camera frame to the pixel frame.

    Args:
        cam_coords: [batch, 4, height,
width]
        proj: [batch, 4, 4]
    Returns:
        Pixel coordinates projected
from the camera frame [batch, height,
width, 2]
    """
    batch = tf.shape(cam_coords)[0]
    height = tf.shape(cam_coords)[2]
    width = tf.shape(cam_coords)[3]

    cam_coords = tf.reshape(cam_coords,
[batch, 4, -1])
    unnormalized_pixel_coords =
tf.matmul(proj, cam_coords)
    x_u =
tf.slice(unnormalized_pixel_coords, [0,
0, 0], [-1, 1, -1])
    y_u =
tf.slice(unnormalized_pixel_coords, [0,
1, 0], [-1, 1, -1])
    z_u =
tf.slice(unnormalized_pixel_coords, [0,
2, 0], [-1, 1, -1])
    x_n = x_u / (z_u + 1e-10)
    y_n = y_u / (z_u + 1e-10)
```

```
    pixel_coords = tf.concat([x_n,
y_n], axis=1)
    pixel_coords =
tf.reshape(pixel_coords, [batch, 2,
height, width])
    reproj_depth = tf.reshape(z_u,
[batch, 1, height, width])
    reproj_depth =
tf.transpose(reproj_depth,
perm=[0,2,3,1])
    return tf.transpose(pixel_coords,
perm=[0, 2, 3, 1]), reproj_depth

def get_visibility_mask(coords):
    """ Get visible region mask
    Args:
        coords: [batch, height, width,
2]
    Return:
        visible mask [batch, height,
width, 1]
    """

    coords_x, coords_y =
tf.split(coords, [1, 1], axis=3)

    coords_x = tf.cast(coords_x,
'float32')
    coords_y = tf.cast(coords_y,
'float32')

    x0 = tf.cast(tf.floor(coords_x),
tf.int32)
    x1 = x0 + 1
    y0 = tf.cast(tf.floor(coords_y),
tf.int32)
    y1 = y0 + 1

    height = tf.shape(coords)[1]
    width = tf.shape(coords)[2]
    zero = tf.zeros([1],
dtype=tf.int32)
    inside_x =
tf.logical_and(tf.greater_equal(x0,
zero), tf.less(x1, width))
    inside_y =
tf.logical_and(tf.greater_equal(y0,
zero), tf.less(y1, height))
    visible_mask =
tf.cast(tf.logical_and(inside_x,
inside_y), tf.float32)

    return visible_mask

def bilinear_sampling(photos, coords,
depths=None):
    """Construct a new image by
bilinear sampling from the input image.

    Points falling outside the source
image boundary have value 0.

    Args:
        photos: source image to be
sampled from [batch, height_s, width_s,
channels]
        coords: coordinates of source
pixels to sample from [batch, height_t,
         width_t, 2]. height_t/width_t
correspond to the dimensions of the
output
         image (don't need to be the
same as height_s/width_s). The two
channels
         correspond to x and y
coordinates respectively.
    Returns:
        A new sampled image [batch,
height_t, width_t, channels]
    """
    # photos: [batch_size, height2,
width2, C]
    # coords: [batch_size, height1,
width1, C]
    # depths: [batch_size, height2,
width2, 1]
    def _repeat(x, n_repeats):
        rep = tf.transpose(

tf.expand_dims(tf.ones(shape=tf.stack([
                n_repeats,
            ])), 1), [1, 0])
        rep = tf.cast(rep, 'float32')
        x = tf.matmul(tf.reshape(x, (-
1, 1)), rep)
        return tf.reshape(x, [-1])

    with
tf.name_scope('image_sampling'):
        coords_x, coords_y =
tf.split(coords, [1, 1], axis=3)
        inp_size = tf.shape(photos)
        if depths is not None:
            dpt_size = tf.shape(depths)
        else:
            dpt_size = inp_size # dammy
        coord_size = tf.shape(coords)

        out_size =
tf.stack([coord_size[0],

coord_size[1],

coord_size[2],

inp_size[3],
                                ])
        out_size_d =
tf.stack([coord_size[0],

coord_size[1],

coord_size[2],

dpt_size[3],
                                ])

        coords_x = tf.cast(coords_x,
'float32')
        coords_y = tf.cast(coords_y,
'float32')

        x0 = tf.floor(coords_x)
        x1 = x0 + 1
        y0 = tf.floor(coords_y)
        y1 = y0 + 1
```

```
        y_max =
tf.cast(tf.shape(photos)[1] - 1,
'float32')
        x_max =
tf.cast(tf.shape(photos)[2] - 1,
'float32')
        zero = tf.zeros([1],
dtype='float32')

        x0_safe = tf.clip_by_value(x0,
zero, x_max)
        y0_safe = tf.clip_by_value(y0,
zero, y_max)
        x1_safe = tf.clip_by_value(x1,
zero, x_max)
        y1_safe = tf.clip_by_value(y1,
zero, y_max)

        ## bilinear interp weights,
with points outside the grid having
weight 0
        # wt_x0 = (x1 - coords_x) *
tf.cast(tf.equal(x0, x0_safe),
'float32')
        # wt_x1 = (coords_x - x0) *
tf.cast(tf.equal(x1, x1_safe),
'float32')
        # wt_y0 = (y1 - coords_y) *
tf.cast(tf.equal(y0, y0_safe),
'float32')
        # wt_y1 = (coords_y - y0) *
tf.cast(tf.equal(y1, y1_safe),
'float32')

        wt_x0 = x1_safe - coords_x
        wt_x1 = coords_x - x0_safe
        wt_y0 = y1_safe - coords_y
        wt_y1 = coords_y - y0_safe

        ## indices in the flat image to
sample from
        dim2 = tf.cast(inp_size[2],
'float32')
        dim1 = tf.cast(inp_size[2] *
inp_size[1], 'float32')
        base = tf.reshape(
            _repeat(

tf.cast(tf.range(coord_size[0]),
'float32') * dim1,
                coord_size[1] *
coord_size[2]),
            [out_size[0], out_size[1],
out_size[2], 1])

        base_y0 = base + y0_safe * dim2
        base_y1 = base + y1_safe * dim2
        idx00 = tf.reshape(x0_safe +
base_y0, [-1])
        idx01 = x0_safe + base_y1
        idx10 = x1_safe + base_y0
        idx11 = x1_safe + base_y1


        ## sample from photos
        photos_flat =
tf.reshape(photos, tf.stack([-1,
inp_size[3]]))
        photos_flat =
tf.cast(photos_flat, 'float32')
        im00 =
tf.reshape(tf.gather(photos_flat,
tf.cast(idx00, 'int32')), out_size)
        im01 =
tf.reshape(tf.gather(photos_flat,
tf.cast(idx01, 'int32')), out_size)
        im10 =
tf.reshape(tf.gather(photos_flat,
tf.cast(idx10, 'int32')), out_size)
        im11 =
tf.reshape(tf.gather(photos_flat,
tf.cast(idx11, 'int32')), out_size)

        w00 = wt_x0 * wt_y0
        w01 = wt_x0 * wt_y1
        w10 = wt_x1 * wt_y0
        w11 = wt_x1 * wt_y1

        out_photos = tf.add_n([
            w00 * im00, w01 * im01,
            w10 * im10, w11 * im11
        ])
        if depths is None:
            return out_photos
        else:
            ## sample from depths
            dpts_flat =
tf.reshape(depths, tf.stack([-1,
dpt_size[3]]))
            dpts_flat =
tf.cast(dpts_flat, 'float32')

            dp00 =
tf.reshape(tf.gather(dpts_flat,
tf.cast(idx00, 'int32')), out_size_d)
            dp01 =
tf.reshape(tf.gather(dpts_flat,
tf.cast(idx01, 'int32')), out_size_d)
            dp10 =
tf.reshape(tf.gather(dpts_flat,
tf.cast(idx10, 'int32')), out_size_d)
            dp11 =
tf.reshape(tf.gather(dpts_flat,
tf.cast(idx11, 'int32')), out_size_d)

            out_depth = tf.add_n([
                w00 * dp00, w01 * dp01,
                w10 * dp10, w11 * dp11
            ])

            return out_photos,
out_depth

def nearest_neighbor_sampling(photos,
coords):
    # photos: [batch_size, height2,
width2, C]
    # coords: [batch_size, height1,
width1, C]
    # outputs: [batch_size, height1,
width1, C]

    inp_size = tf.shape(photos)
    coord_size = tf.shape(coords)
    out_size = tf.stack([coord_size[0],
                         coord_size[1],
                         coord_size[2],
                         inp_size[3],
                        ])
```

```
    batch_size = inp_size[0]
    in_height = inp_size[1]
    in_width = inp_size[2]
    out_height = out_size[1]
    out_width = out_size[2]

    coords_x, coords_y =
tf.split(coords, [1, 1], axis=3)

    # [height+2,width+2]
    photos_pad = tf.pad(photos, [[0,
0], [1, 1], [1, 1], [0,0]], 'CONSTANT')

    # valid area = 1 <= x,y <= width
    coords_x_pad =
tf.clip_by_value(tf.cast(tf.round(coord
s_x)+1, tf.int32), 0, in_width+1)
    coords_y_pad =
tf.clip_by_value(tf.cast(tf.round(coord
s_y)+1, tf.int32), 0, in_height+1)

    batch_inds =
tf.tile(tf.range(batch_size)[:,None,Non
e,None], [1, out_height, out_width,1])
    byx = tf.concat([batch_inds,
coords_y_pad, coords_x_pad], axis=-1)

    outputs = tf.gather_nd(photos_pad,
byx)
    return outputs

# def projective_inverse_warp(photos1,
depths2, c1Tc2s, intrinsics_3x3,
depths1, depth_thresh=1.0,
name='InvWarp'):
#     '''
#     Args:
#         photos1 : [B,H,W,C] photos of
camera1
#         depths2 : [B,H,W,C] depths of
camera2
#         c1Tc2s : [B,4,4] pose matrix
from camera2 to camera1
#         intrinsics_3x3 : [3,3]
intrinsic matrix
#         depths1 : [B,H,W,C] (option)
depths of camera1. depths1 is used to
remove occlusion from valid_masks
#         depth_thresh : float (option)
threshold for depth misconsistency
between depths2 and warped depths
#     '''
#     with tf.name_scope(name):
#         batch = tf.shape(photos1)[0]
#         height = tf.shape(photos1)[1]
#         width = tf.shape(photos1)[2]

#         inv_intrinsics_3x3 =
tf.matrix_inverse(intrinsics_3x3)
#         batch_intrinsics_4x4 =
tf.concat([intrinsics_3x3,
tf.zeros([3,1])], axis=1)
#         batch_intrinsics_4x4 =
tf.concat([batch_intrinsics_4x4,
tf.constant([0.0,0.0,0.0,1.0],
shape=[1,4])], axis=0)
#         batch_intrinsics_4x4 =
tf.expand_dims(batch_intrinsics_4x4,
axis=0) # [1,4,4]
#         batch_intrinsics_4x4 =
tf.tile(batch_intrinsics_4x4, [batch,
1, 1]) # [B,4,4]

#         pixel_coords_c2 =
meshgrid(batch, height, width)
#         cam_coords_c2 =
pixel2cam(depths2, pixel_coords_c2,
inv_intrinsics_3x3)
#         proj_c1Tc2s =
tf.matmul(batch_intrinsics_4x4, c1Tc2s)
#         pixel_coords_c1,
reproj_depths =
cam2pixel(cam_coords_c2, proj_c1Tc2s)

#         warp_photos, warp_depths =
bilinear_sampling(photos1,
pixel_coords_c1, depths1)

#
warp_photos.set_shape(photos1.shape)
#
warp_depths.set_shape(depths1.shape)

#         # visiblity mask
#         visible_masks =
get_visibility_mask(pixel_coords_c1)
#         camfront_masks =
tf.cast(tf.greater(reproj_depths,
tf.zeros((), reproj_depths.dtype)),
tf.float32)
#         nonocc_masks =
tf.cast(tf.less(tf.squared_difference(w
arp_depths, depths2), depth_thresh**2),
tf.float32)
#         visible_masks = visible_masks
* camfront_masks * nonocc_masks # take
logical_and
#         warp_photos = warp_photos *
visible_masks

#     return warp_photos, visible_masks

def projective_inverse_warp(photos1,
depths2, c1Tc2s, intrinsics_3x3,
depths1, depth_thresh=1.0,
name='InvWarp'):
    '''
    Args:
        photos1 : [B,H,W,C] photos of
camera1
        depths2 : [B,H,W,C] depths of
camera2
        c1Tc2s : [B,4,4] pose matrix
from camera2 to camera1
        intrinsics_3x3 : [B,3,3]
intrinsic matrix
        depths1 : [B,H,W,C] (option)
depths of camera1. depths1 is used to
remove occlusion from valid_masks
        depth_thresh : float (option)
threshold for depth misconsistency
between depths2 and warped depths
    '''
    with tf.name_scope(name):
        batch = tf.shape(photos1)[0]
        height = tf.shape(photos1)[1]
        width = tf.shape(photos1)[2]
```

```
        inv_intrinsics_3x3 =
tf.matrix_inverse(intrinsics_3x3)


        # if thetas1 is not None:
        #     inv_thetas1 =
tf.matrix_inverse(thetas1)
        #     inv_intrinsics_3x3 =
tf.matmul(inv_thetas1,
inv_intrinsics_3x3)

        right_col =
tf.zeros([batch,3,1], dtype=tf.float32)
        bottom_row =
tf.tile(tf.constant([0,0,0,1],
dtype=tf.float32, shape=[1,1,4]),
[batch,1,1])
        batch_intrinsics_4x4 =
tf.concat([intrinsics_3x3, right_col],
axis=2)
        batch_intrinsics_4x4 =
tf.concat([batch_intrinsics_4x4,
bottom_row], axis=1)

        pixel_coords_c2 =
meshgrid(batch, height, width)
        cam_coords_c2 =
pixel2cam(depths2, pixel_coords_c2,
inv_intrinsics_3x3)
        proj_c1Tc2s =
tf.matmul(batch_intrinsics_4x4, c1Tc2s)
        pixel_coords_c1, reproj_depths
= cam2pixel(cam_coords_c2, proj_c1Tc2s)

        warp_photos, warp_depths =
bilinear_sampling(photos1,
pixel_coords_c1, depths1)


warp_photos.set_shape(photos1.shape)

warp_depths.set_shape(depths1.shape)

        # visiblity mask
        visible_masks =
get_visibility_mask(pixel_coords_c1)
        camfront_masks =
tf.cast(tf.greater(reproj_depths,
tf.zeros((), reproj_depths.dtype)),
tf.float32)
        nonocc_masks =
tf.cast(tf.less(tf.squared_difference(w
arp_depths, depths2), depth_thresh**2),
tf.float32)
        visible_masks = visible_masks *
camfront_masks * nonocc_masks # take
logical_and
        warp_photos = warp_photos *
visible_masks

    return warp_photos, visible_masks
def extract_xy_coords(d_heatmaps,
block_size):
    batch = tf.shape(d_heatmaps)[0]
    rheight = tf.shape(d_heatmaps)[1]
    rwidth = tf.shape(d_heatmaps)[2]
    width = rwidth * block_size
    height = rheight * block_size
    d_argmax =
tf.cast(tf.argmax(d_heatmaps, axis=-1),
dtype=tf.int32)
    fgmask =
tf.cast(tf.not_equal(d_argmax,
block_size**2), dtype=tf.int32)

    x_bcoords = tf.mod(d_argmax,
block_size)
    y_bcoords = tf.floordiv(d_argmax,
block_size) # floor_div ?
    zero = tf.constant(0,
dtype=tf.int32)
    zeros = tf.zeros_like(x_bcoords)

    x_bcoords =
tf.where(tf.equal(fgmask, zero), zeros,
x_bcoords)
    y_bcoords =
tf.where(tf.equal(fgmask, zero), zeros,
y_bcoords)

    x_offset, y_offset =
tf.meshgrid(tf.range(0, width,
block_size), tf.range(0, height,
block_size))
    x_offset =
tf.tile(tf.expand_dims(x_offset,
axis=0), [batch, 1, 1])
    y_offset =
tf.tile(tf.expand_dims(y_offset,
axis=0), [batch, 1, 1])

    x_icoords = x_bcoords + x_offset
    y_icoords = y_bcoords + y_offset

    return x_icoords, y_icoords, fgmask


def
heatmaps_to_reprojected_heatmaps(d_heat
maps1, depths1, depths2, c2Tc1s,
intrinsics, block_size,
depth_thresh=1.0, name='HM2HM'):

    '''
    convert CNN output d_heatmaps to
reprojected heatmaps

    Args:
        d_heatmaps1: [batch, height/N,
width/N, N*N+1], CNN output depth-wise
heatmaps
        depths1: [batch, height, width,
1], depths of photos1
        depths2: [batch, height, width,
1], depths of photos2 (to check depth
consistency)
        c2Tc1s: [batch, 4, 4],
transformation matrix from camera1 to
camera2
        intrinsics: [batch, 3, 3],
intrinsics matrix
        block_size: N, downsampling
rate (2**#pooling)
    Return:
        heatmaps2: [batch, height,
width, 1]
    '''
```

```
    with tf.name_scope(name):
        batch = tf.shape(depths1)[0]
        height = tf.shape(depths1)[1]
        width = tf.shape(depths1)[2]

        rheight =
tf.shape(d_heatmaps1)[1]
        rwidth =
tf.shape(d_heatmaps1)[2]

        # extract keypoints from each
blocks
        with tf.name_scope('KP-
EXTRACT'):
            x1, y1, fgmask =
extract_xy_coords(d_heatmaps1,
block_size)
            fgmask = tf.reshape(fgmask,
(batch, 1, -1))
        with tf.name_scope('XY2IDX'):
            # convert (x,y) to
idx=x+y*width+b*width*height
            x1 = tf.reshape(x1, (batch,
1, -1))
            y1 = tf.reshape(y1, (batch,
1, -1))
            b_offset = tf.range(batch)
* width * height
            b_offset =
tf.tile(tf.expand_dims(b_offset,
axis=1), [1, rwidth*rheight])
            b_offset =
tf.reshape(b_offset, (batch, 1, -1))
            idx1 = x1 + y1 * width +
b_offset
            # extract depth on (x,y)
            depth1s_flt =
tf.reshape(depths1, [-1])
            z1 = tf.gather(depth1s_flt,
idx1)

        with tf.name_scope('PIX2CAM'):

            # pixel to camera
coordinate (x,y,1 --> x,y,z,1)
            ones = tf.ones_like(x1)
            pix_coords = tf.concat([x1,
y1, ones], axis=1)
            pix_coords =
tf.cast(pix_coords, tf.float32)

            inv_intrinsics =
tf.matrix_inverse(intrinsics)
            cam_coords =
tf.matmul(inv_intrinsics, pix_coords) *
z1
            cam_coords =
tf.concat([cam_coords,
tf.cast(ones,tf.float32)], axis=1)

        with tf.name_scope('TRANS-
MAT'):
            # get Transform matrix
            filler = tf.constant([0.0,
0.0, 0.0, 1.0], shape=[1, 1, 4])
            filler = tf.tile(filler,
[batch, 1, 1])
            intrinsics_4x4 =
tf.concat([intrinsics, tf.zeros([batch,
3, 1])], axis=2)
            intrinsics_4x4 =
tf.concat([intrinsics_4x4, filler],
axis=1)
            proj_mats =
tf.matmul(intrinsics_4x4, c2Tc1s)

        with tf.name_scope('CAM2PIX'):
            # camera to pixel
coordinate
            unnormalized_pixel_coords =
tf.matmul(proj_mats, cam_coords) #
[batch, 4, N(=rwidth*rheight)]

            x_u =
tf.slice(unnormalized_pixel_coords,
[0,0,0], [-1,1,-1])
            y_u =
tf.slice(unnormalized_pixel_coords,
[0,1,0], [-1,1,-1])
            z_u =
tf.slice(unnormalized_pixel_coords,
[0,2,0], [-1,1,-1])
            # remove coordinate behind
camera
            cam_front = tf.greater(z_u,
0)
            fgmask = fgmask *
tf.cast(cam_front, tf.int32)
            epsilon = tf.constant(1e-6,
dtype=tf.float32)
            x_n = x_u / (z_u + epsilon)
            y_n = y_u / (z_u + epsilon)

        with
tf.name_scope('DepthCheck'):
            x2 =
tf.clip_by_value(tf.cast(tf.round(x_n),
tf.int32), 0, width-1)
            y2 =
tf.clip_by_value(tf.cast(tf.round(y_n),
tf.int32), 0, height-1)
            depth2s_flt =
tf.reshape(depths2, [-1])
            idx2 = x2 + y2 * width +
b_offset
            z2 = tf.gather(depth2s_flt,
idx2)
            near_depth =
tf.cast(tf.less_equal(tf.abs(z_u-z2),
depth_thresh), tf.int32)
            fgmask = fgmask *
near_depth

        with
tf.name_scope('SUBSAMPLING'):
            # clipping coordinates
            pad = 1
            width2 = width + 2 * pad
            height2 = height + 2 * pad

            x2 =
tf.clip_by_value(tf.cast(tf.round(x_n),
tf.int32)+pad, 0, width+1) # 0~width+1
(actual range=1~width-1)
            y2 =
tf.clip_by_value(tf.cast(tf.round(y_n),
tf.int32)+pad, 0, height+1)
            zero = tf.constant(0,
dtype=tf.int32)
            zeros = tf.zeros_like(x2)
```

```
            x2 =
tf.where(tf.equal(fgmask, zero), zeros,
x2)
            y2 =
tf.where(tf.equal(fgmask, zero), zeros,
y2)
            x2 = tf.reshape(x2, [batch,
-1])
            y2 = tf.reshape(y2, [batch,
-1])

            b_offset = tf.range(batch)
* width2 * height2
            b_offset =
tf.tile(tf.expand_dims(b_offset,
axis=1), [1, rwidth*rheight])

            idx2 = x2 + y2 * width2 +
b_offset
            idx2 = tf.reshape(idx2, [-
1])

            idx2, _ = tf.unique(idx2) #
remove duplicate indices to apply
sparce_to_dense

            # ignore sorted order
(https://stackoverflow.com/questions/35
508894/sparse-to-dense-requires-
indices-to-be-lexicographically-sorted-
in-0-7)
            heatmaps2 =
tf.sparse_to_dense(idx2,
[batch*height2*width2], 1, 0,
validate_indices=False)

            heatmaps2 =
tf.reshape(heatmaps2, (batch, height2,
width2))
            heatmaps2 =
tf.slice(heatmaps2, [0,1,1], [-
1,height,width])
            heatmaps2 =
tf.cast(tf.expand_dims(heatmaps2,
axis=-1), tf.float32)

    return heatmaps2


# def extract_xy_coords(d_heatmaps,
block_size):
#     batch = tf.shape(d_heatmaps)[0]
#     rheight = tf.shape(d_heatmaps)[1]
#     rwidth = tf.shape(d_heatmaps)[2]
#     width = rwidth * block_size
#     height = rheight * block_size

#     d_argmax =
tf.cast(tf.argmax(d_heatmaps, axis=-1),
dtype=tf.int32)
#     fgmask =
tf.cast(tf.not_equal(d_argmax,
block_size**2), dtype=tf.int32)

#     x_bcoords = tf.mod(d_argmax,
block_size)
#     y_bcoords = tf.floordiv(d_argmax,
block_size) # floor_div ?
#     zero = tf.constant(0,
dtype=tf.int32)
#     zeros = tf.zeros_like(x_bcoords)

#     x_bcoords =
tf.where(tf.equal(fgmask, zero), zeros,
x_bcoords)
#     y_bcoords =
tf.where(tf.equal(fgmask, zero), zeros,
y_bcoords)

#     x_offset, y_offset =
tf.meshgrid(tf.range(0, width,
block_size), tf.range(0, height,
block_size))
#     x_offset =
tf.tile(tf.expand_dims(x_offset,
axis=0), [batch, 1, 1])
#     y_offset =
tf.tile(tf.expand_dims(y_offset,
axis=0), [batch, 1, 1])

#     x_icoords = x_bcoords + x_offset
#     y_icoords = y_bcoords + y_offset

#     # batch offset
#     batch_offset = tf.range(batch) *
width * height
#     batch_offset =
tf.tile(tf.expand_dims(batch_offset,
axis=1), [1, rwidth*rheight])
#     batch_offset =
tf.reshape(batch_offset, shape=(batch,
rheight, rwidth))

#     return x_icoords, y_icoords,
batch_offset, fgmask # [batch, rheight,
rwidth]


# def
heatmaps_to_reprojected_heatmaps(src_d_
heatmaps, depth1s, c2Tc1s, intrinsics,
block_size, name='HM2HM'):

#     '''
#     convert CNN output d_heatmaps to
reprojected heatmaps

#     Args:
#         src_d_heatmaps: [batch,
height/N, width/N, N*N+1], CNN output
#         depths1s: [batch, height,
width, 1], depths of photos1
#         c2Tc1s: [batch, 4, 4],
transformation matrix from camera1 to
camera2
#         intrinsics: [batch, 3, 3],
intrinsics matrix
#         block_size: N, downsampling
rate (2**#pooling)
#     Return:
#         dst_heatmaps: [batch, height,
width, 1]
#     '''
#     with tf.name_scope(name):
#         batch = tf.shape(depth1s)[0]
#         height = tf.shape(depth1s)[1]
#         width = tf.shape(depth1s)[2]

#         rheight =
tf.shape(src_d_heatmaps)[1]
```

```
#         rwidth =
tf.shape(src_d_heatmaps)[2]

#         # extract keypoints from each
blocks
#         with tf.name_scope('KP-
EXTRACT'):
#             x_src, y_src, b_offset,
fgmask =
extract_xy_coords(src_d_heatmaps,
block_size)


#         with tf.name_scope('XY2IDX'):
#             # convert (x,y) to
idx=x+y*width+b*width*height
#             x_src = tf.reshape(x_src,
(batch, 1, -1))
#             y_src = tf.reshape(y_src,
(batch, 1, -1))
#             b_offset =
tf.reshape(b_offset, (batch, 1, -1))
#             idx_src = x_src + y_src *
width + b_offset

#         with
tf.name_scope('PIX2CAM'):
#             # extract depth on (x,y)
#             depth1s_flt =
tf.reshape(depth1s, [-1])
#             z1_src =
tf.gather(depth1s_flt, idx_src)

#             # pixel to camera
coordinate (x,y,1 --> x,y,z,1)
#             ones =
tf.ones_like(x_src)
#             pix_coords =
tf.concat([x_src, y_src, ones], axis=1)
#             pix_coords =
tf.cast(pix_coords, tf.float32)

#             inv_intrinsics =
tf.matrix_inverse(intrinsics)
#             cam_coords =
tf.matmul(inv_intrinsics, pix_coords) *
z1_src
#             cam_coords =
tf.concat([cam_coords,
tf.cast(ones,tf.float32)], axis=1)


#         with tf.name_scope('TRANS-
MAT'):
#             # get Transform matrix
#             filler =
tf.constant([0.0, 0.0, 0.0, 1.0],
shape=[1, 1, 4])
#             filler = tf.tile(filler,
[batch, 1, 1])
#             intrinsics_4x4 =
tf.concat([intrinsics, tf.zeros([batch,
3, 1])], axis=2)
#             intrinsics_4x4 =
tf.concat([intrinsics_4x4, filler],
axis=1)
#             proj_mats =
tf.matmul(intrinsics_4x4, c2Tc1s)

#         with
tf.name_scope('CAM2PIX'):
#             # camera to pixel
coordinate
#             unnormalized_pixel_coords
= tf.matmul(proj_mats, cam_coords) #
[batch, 4, N(=rwidth*rheight)]

#             x_u =
tf.slice(unnormalized_pixel_coords,
[0,0,0], [-1,1,-1])
#             y_u =
tf.slice(unnormalized_pixel_coords,
[0,1,0], [-1,1,-1])
#             z_u =
tf.slice(unnormalized_pixel_coords,
[0,2,0], [-1,1,-1])
#             epsilon = tf.constant(1e-
10, dtype=tf.float32)
#             x_n = x_u / (z_u +
epsilon)
#             y_n = y_u / (z_u +
epsilon)

#         with
tf.name_scope('SUBSAMPLING'):
#             # clipping coordinates
#             pad = 1
#             width2 = width + 2 * pad
#             height2 = height + 2 *
pad
#             x_n =
tf.clip_by_value(tf.cast(tf.round(x_n),
tf.int32)+pad, 0, width+1) # 0~width+1
(actual range=1~width-1)
#             y_n =
tf.clip_by_value(tf.cast(tf.round(y_n),
tf.int32)+pad, 0, height+1)
#             zero = tf.constant(0,
dtype=tf.int32)
#             zeros =
tf.zeros_like(x_n)
#             fgmask =
tf.reshape(fgmask, (batch, 1, -1))
#             x_n =
tf.where(tf.equal(fgmask, zero), zeros,
x_n)
#             y_n =
tf.where(tf.equal(fgmask, zero), zeros,
y_n)
#             x_n = tf.reshape(x_n,
[batch, -1])
#             y_n = tf.reshape(y_n,
[batch, -1])

#             b_offset =
tf.range(batch) * width2 * height2
#             b_offset =
tf.tile(tf.expand_dims(b_offset,
axis=1), [1, rwidth*rheight])

#             idx_dst = x_n + y_n *
width2 + b_offset
#             idx_dst =
tf.reshape(idx_dst, [-1])

#             idx_dst, _ =
tf.unique(idx_dst) # remove duplicate
indices to apply sparce_to_dense
```

```
#             # ignore sorted order
(https://stackoverflow.com/questions/35
508894/sparse-to-dense-requires-
indices-to-be-lexicographically-sorted-
in-0-7)
#             dst_heatmaps =
tf.sparse_to_dense(idx_dst,
[batch*height2*width2], 1, 0,
validate_indices=False)


#             dst_heatmaps =
tf.reshape(dst_heatmaps, (batch,
height2, width2))
#             dst_heatmaps =
tf.slice(dst_heatmaps, [0,1,1], [-
1,height,width])
#             dst_heatmaps =
tf.cast(tf.expand_dims(dst_heatmaps,
axis=-1), tf.float32)

#         return dst_heatmaps

def compute_multi_gradients(photos,
pad, name='SMOOTHNESS'):
    with tf.name_scope(name):
        height = tf.shape(photos)[1]
        width = tf.shape(photos)[2]
        Dx = tf.zeros_like(photos)
        Dy = tf.zeros_like(photos)
        photos_pad = tf.pad(photos,
[[0, 0], [pad, pad], [pad, pad],
[0,0]], 'REFLECT')

        for i in range(pad):
            s = pad + i + 1
            Dx = Dx +
tf.slice(photos_pad, [0,pad,s,0], [-
1,height,width,-1])\
                -
tf.slice(photos_pad, [0,pad,i,0], [-
1,height,width,-1])
            Dy = Dy +
tf.slice(photos_pad, [0,s,pad,0], [-
1,height,width,-1])\
                -
tf.slice(photos_pad, [0,i,pad,0], [-
1,height,width,-1])
        Da = tf.sqrt(Dx**2+Dy**2)

    return Dx, Dy, Da

def
compute_fg_mask_from_gradients(gradient
s, block_size, grad_thresh,
reduce_op=tf.reduce_mean,
name='FGMASK', keep_dims=False):

    with tf.name_scope(name):
        d_grads =
tf.space_to_depth(gradients,
block_size)
        d_grads = reduce_op(d_grads,
axis=3, keep_dims=keep_dims)
        d_fgmask =
tf.cast(tf.greater(d_grads,
grad_thresh), tf.float32)

        # restore fgmask to original
resolution
        # d_fgmask2 =
tf.tile(tf.expand_dims(d_fgmask, -1),
[1,1,1,block_size**2])
        # fgmask =
tf.depth_to_space(d_fgmask2,
block_size)

        return d_fgmask

def compute_background_loss(d_heatmaps,
d_fgmask, name='BG-LOSS'):
    '''
    Args:
        d_heatmaps: [batch, height/N,
width/N, N**2+1], heatmap logits
        d_fgmask: [batch,
height/N,width/N,1], 0/1 mask
    '''
    with tf.name_scope(name):
        # fg_logits, bg_logits =
tf.split(d_heatmaps, [-1,1], axis=-1)
        # d_bgmask = 1.0 - d_fgmask

        # I wonder
sigmoid_cross_entropy is fine because
d_heatmaps is a malti-class tensor
originally
        # xentropy =
tf.nn.sigmoid_cross_entropy_with_logits
(labels=d_bgmask, logits=bg_logits) #
shape is the same as inputs
[batch,height/N, width/N, 1]
        # bg_count = tf.maximum(1.0,
tf.reduce_sum(d_bgmask))
        # bg_loss =
tf.div(tf.reduce_sum(d_bgmask *
xentropy), bg_count) # not take whole
mean

        # neumerically unstable way
        #
https://www.tensorflow.org/get_started/
mnist/beginners
        # fg_probs, bg_probs =
tf.split(d_heatmaps, [-1,1], axis=-1)
        d_bgmask = 1.0 - d_fgmask

        # (1) tf sigmoid xentropy impl.
        # bg_loss =
tf.reduce_mean(tf.nn.sigmoid_cross_entr
opy_with_logits(labels=d_bgmask,
logits=d_heatmaps))
        # (2) naive sigmoid xentropy
impl.
        # bg_prob =
tf.nn.sigmoid(d_heatmaps)
        # xentropy = -d_bgmask *
tf.log(bg_prob) - (1-d_bgmask) *
tf.log(1.0-bg_prob) # - y_label *
log(y_pred)
        # bg_loss =
tf.reduce_mean(xentropy)

        # fg_mask = 1.0 -
tf.nn.sigmoid(d_heatmaps)

        # (3) tf softmax xentropy impl
        # import IPython
        # IPython.embed()
```

```
        d_prob =
tf.nn.softmax(d_heatmaps) # normalize
        fg_prob, bg_prob =
tf.split(d_prob, [-1,1], axis=-1)
        print('Map shape ',
d_heatmaps.shape, d_prob.shape,
bg_prob.shape)
        eps = 1e-6
        bg_prob =
tf.clip_by_value(bg_prob, eps, 1.0-eps)
# to avoid NaN at tf.log
        xentropy = -d_bgmask *
tf.log(bg_prob) - (1-d_bgmask) *
tf.log(1.0-bg_prob) # - y_label *
log(y_pred)
        bg_loss =
tf.reduce_mean(xentropy)
        fg_mask = 1.0 - bg_prob

        # bg_count = tf.maximum(1.0,
tf.reduce_sum(d_bgmask))
        # bg_loss =
tf.div(tf.reduce_sum(d_bgmask *
xentropy), bg_count) # not take whole
mean

        rheight = tf.shape(fg_mask)[1]
        rwidth = tf.shape(fg_mask)[2]
        block_size = 16
        fg_mask =
tf.image.resize_images( fg_mask,
(rheight*block_size,
rwidth*block_size),
method=tf.image.ResizeMethod.NEAREST_NE
IGHBOR)

tf.summary.image('pred_fgmask2',
fg_mask, max_outputs=4)

        return bg_loss

def
compute_repeatable_loss(pred_d_heatmaps
, trans_heatmaps, block_size,
name='REPEAT-LOSS'):
    '''
    Args:
        pred_d_heatmaps: [batch,
height/N, width/N, N**2+1]
        trans_heatmaps: [batch, height,
width, 1]
    '''
    with tf.name_scope(name):
        trans_d_heatmaps =
tf.space_to_depth(trans_heatmaps,
block_size)
        kp_bg_map =
tf.reduce_sum(trans_d_heatmaps, axis=-
1, keep_dims=True)
        kp_bg_map =
tf.cast(tf.less(kp_bg_map, 1.0),
tf.float32)
        kp_fg_map = 1.0 -
tf.squeeze(kp_bg_map, axis=-1)

        trans_d_heatmaps =
tf.concat([trans_d_heatmaps,
kp_bg_map], axis=3) # add BG channels

        xentropy =
tf.nn.softmax_cross_entropy_with_logits
(labels=trans_d_heatmaps,

logits=pred_d_heatmaps) # shape =
[batch,height/N,width/N]
        kp_count = tf.maximum(1.0,
tf.reduce_sum(kp_fg_map))
        repeat_loss =
tf.div(tf.reduce_sum(kp_fg_map *
xentropy), kp_count)

        return repeat_loss

def get_R_loss(pred_d_heatmaps,
trans_heatmaps, d_fgmask, weight_bg,
block_size):

    # repeat_loss =
compute_repeatable_loss(pred_d_heatmaps
, trans_heatmaps, block_size, name='R-
Loss')
    bg_loss =
compute_background_loss(pred_d_heatmaps
, d_fgmask, name='BG-Loss')

    # loss = repeat_loss + weight_bg *
bg_loss
    loss = bg_loss

    # tf.summary.scalar('repeat_loss',
repeat_loss)
    # tf.summary.scalar('bg_loss',
bg_loss)
    tf.summary.scalar('loss', loss)

    return loss

def soft_max_and_argmax_1d(inputs,
axis=-1, inputs_index=None,
keep_dims=False, com_strength1=250.0,
com_strength2=250.0):

    # Safe softmax
    inputs_exp1 =
tf.exp(com_strength1*(inputs -
tf.reduce_max(inputs, axis=axis,
keep_dims=True)))
    inputs_softmax1 = inputs_exp1 /
(tf.reduce_sum(inputs_exp1, axis=axis,
keep_dims=True) + 1e-8)

    inputs_exp2 =
tf.exp(com_strength2*(inputs -
tf.reduce_max(inputs, axis=axis,
keep_dims=True)))
    inputs_softmax2 = inputs_exp2 /
(tf.reduce_sum(inputs_exp2, axis=axis,
keep_dims=True) + 1e-8)

    inputs_max = tf.reduce_sum(inputs *
inputs_softmax1, axis=axis,
keep_dims=keep_dims)

    inputs_index_shp =
[1,]*len(inputs.get_shape())
    inputs_index_shp[axis] = -1
    if inputs_index is None:
        inputs_index =
tf.range(inputs.get_shape().as_list()[a
```

```
xis], dtype=inputs.dtype) # use
0,1,2,..,inputs.shape[axis]-1
    inputs_index =
tf.reshape(inputs_index,
inputs_index_shp)
    inputs_amax =
tf.reduce_sum(inputs_index *
inputs_softmax2, axis=axis,
keep_dims=keep_dims)

    return inputs_max, inputs_amax

def soft_argmax_2d(patches_bhwc,
patch_size, do_softmax=True,
com_strength=10):
    # Returns the relative soft-argmax
position, in the -1 to 1 coordinate
    # system of the patch

    width = patch_size
    height = patch_size

    x_t =
tf.matmul(tf.ones(shape=tf.stack([heigh
t, 1])),

tf.transpose(tf.expand_dims(tf.linspace
(-1.0, 1.0, width), 1), [1, 0]))
    y_t =
tf.matmul(tf.expand_dims(tf.linspace(-
1.0, 1.0, height), 1),

tf.ones(shape=tf.stack([1, width])))
    xy_grid = tf.stack([x_t, y_t],
axis=-1)[None] # BHW2

    maxes_bhwc = patches_bhwc
    if do_softmax:
        exps_bhwc = tf.exp(

com_strength*(patches_bhwc -
tf.reduce_max(

patches_bhwc, axis=(1, 2),
keep_dims=True)))
        maxes_bhwc = exps_bhwc / (
            tf.reduce_sum(exps_bhwc,
axis=(1, 2), keep_dims=True) + 1e-8)

    dxdy = tf.reduce_sum(xy_grid *
maxes_bhwc, axis=(1,2))

    return dxdy
```